Three-Dimensional Volumetric Scene Recovery from Multiple Stereo Views Using Voxel Division Techniques

Undergraduate Honors Thesis

Presented in Partial Fulfillment of the Requirements for

Graduation with Distinction

at The Ohio State University

By

Jordan David Lawver

Undergraduate Program in Geomatics Engineering

The Ohio State University

2011

Thesis Committee:

Dr. Alper Yilmaz, Advisor

Dr. Rongxing Li

ABSTRACT

Traditionally, the reconstruction of three-dimensional (3D) scene models from multiple stereo images has required a set of matching feature points, such as building corners, across multiple viewpoints. These points are back-projected to triangulate at 3D positions (voxels) denoting their position in 3D space. These voxels are then used to generate 3D surfaces, thus creating a sparse 3D model in the object space. The sparsity of recovered voxels is an undesired byproduct of triangulation-based methods; an issue addressed using more recent volumetric techniques. As opposed to their triangulation-based counterparts, volumetric methods generate a dense 3D model and reduce the ambiguities in matching and triangulating points. Volumetric methods project all voxels from a grid in 3D space to each image plane using known camera parameters. Projected voxels containing similar color across multiple images are labeled opaque such that they lie on the 3D surface, whereas those voxels projecting to different colors are labeled transparent, representing emptiness in 3D space.

The processing time for volumetric methods is considerably higher than triangulation-based methods, due to traversing all possible voxels on a dense 3D grid. Current published literature on volumetric methods primarily

concentrates on metrics defining similarity of colors and visibility tests without consideration for high processing time requirements. In this research, this limitation has been addressed by employing a "divide and conquer" voxel-division algorithm. This approach initially assumes the bounding box around the object in the 3D scene as a single voxel projecting to the corresponding region across all stereo views. The similarity of visual features computed from the regions across the images is used to determine whether a higher granularity (i.e. dividing the current voxel into eight subvoxels) is necessary to accurately represent that area in object space. This recursive division is continued until an accurate representation of the voxel can be determined. Preliminary results show a reduction in processing time as much as 97% in some tests without a loss in recovered 3D quality. With this improvement, many deficiencies in existing practical applications using volumetric methods can be overcome, including but not limited to automated surveillance in 3D space, biomechanics research, and remotely monitored e-ICUs (electronic intensive care units).

ACKNOWLEDGMENTS

I would like to first and foremost thank my advisor, Professor Alper Yilmaz, for giving me the opportunity for an incredible learning opportunity through my undergraduate research project as well as the abundance of knowledge and support he has lent along the way. I would also like to thank Professor Rongxing Li for serving on my defense committee, as well as the extensive learning opportunities he provided me while working at the Mapping & GIS Lab. In addition, I would especially like to thank Michael Mason and the College of Engineering Honors and Scholars Department for their generous scholarship and continued assistance in helping me complete my research project. I would also like to thank fellow students Kevin Cantwell, Justin Crawford, Daniya Zamalieva, Andrew Kerns, and Boris Skopljak for their assisted knowledge and support for my research topic. Finally, I would like to give my utmost appreciation to my wonderful fiancée, family, and friends for whom completing this endeavor would have been impossible without.

VITA

May 2006 . . . . . Indian Valley High School

2008 to present . . . . . Undergraduate Research Assistant,
Department of CEEGS,
The Ohio State University

2011 . . . . . B.S. Geomatics Engineering,
The Ohio State University

Fields of Study

Major Field: Geomatics Engineering

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

1.1 Problem

Three-dimensional scene recovery is the process used to convert multiple two-dimensional stereo images of an object into a single three-dimensional model. Until recently, three-dimensional scene recovery has been done exclusively using methods based on feature matching. This process involves using complex algorithms to determine a large number of matching points over many stereo images and subsequently back-projecting and triangulating to 3D locations (voxels) to reconstruct the three-dimensional object [1].

While typically effective, many disadvantages come along with this approach. First, extracting and matching feature points is a very complex task. Determining matching features requires advanced mathematical algorithms that typically still result in mismatched points. In addition, since features are not defined on a sub-pixel level and are generalized to the pixel center, matched feature points typically do not exactly correspond to each other [3]. Second, multiple stereo views must be generally close together (i.e., small camera baseline) so that features look similar across multiple views. With a large camera baseline, alike points are skewed and shadowed differently and resultantly more difficult to match.

Next, the integer-valued matched points and corresponding camera parameters typically result in non-converging back-projection lines in three-dimensional space [3]. As such, a bundle adjustment is needed to adjust the camera parameters and matched points [1]. This is a difficult task that can result in inaccurate correspondences and triangulations. Fourth, correspondences between camera parameters and matched points must be maintained over many views spanning large viewpoint changes. As the number of stereo views increases, as does the complexity of the system. This issue has been tackled in mathematical modeling, but typically results in complex time-consuming computations and resulting errors. Finally, except for in rare cases where you can successfully extract a dense amount of matched points, one is typically left with a sparse set of matched features resulting in a sparse scene recovery [1]. In this situation, a parameterized surface model must be estimated to achieve a dense surface reconstruction, a process that adds even more estimation to the final product.

To correct the issues in traditional image-based reconstruction methods, researchers have introduced three-dimensional volumetric scene recovery. As opposed to triangulation-based methods, volumetric scene recovery uses forward projections along a dense grid of voxels in the entire three-dimensional object space such that no surface features are left undetermined [9]. This method projects all voxels on the grid to their respective regions on each image plane using known camera parameters. Projected voxels containing similar color across multiple images are labeled opaque such that they lie on the 3D surface, whereas those voxels projecting to different colors are labeled transparent, representing emptiness in 3D space. The primary benefit of three-dimensional volumetric scene

recovery is in the fact that all computation is done in the three-dimensional object space, such that no triangulation is necessary. Though more effective than traditional approaches, this new method is significantly bottlenecked in computational speed. Volumetric scene recovery creates a highly granular grid of voxels in the object space and thus requires every voxel to be thoroughly analyzed. Unfortunately, this approach requires unnecessary calculations for 3D scene voxels that do not correspond to any object.

The voxel division techniques employed in my research seek to erase this issue by performing a coarse-to-fine recovery using the "divide-and-conquer" methodology. As opposed to beginning with a highly granular grid of voxels, the algorithm first analyzes the entire 3D object's bounding box as a single voxel. The voxel division methodology then recursively divides voxels into 2x2x2 sets of eight subvoxels. As the recursion progresses, each newly created voxel is analyzed for similarities in the image space to determine if its contents are lying on or off the object surface. Once a voxel is determined to be on or off the object surface, no further division is required, as it is known that any further division will result in the same classification. In the traditional volumetric scene recovery approach, all of these granular sub-voxels would have required independent analysis, increasing processing time only to achieve the same result.

It is the aim of this thesis to create an efficient algorithm for reconstructing three-dimensional objects that can be applied universally to all existing volumetric methods. It is apparent that triangulation-based methods lack quality and traditional volumetric

methods lack speed, so it is within the context of my research that I hope to contribute a method that can successfully fulfill both.

1.2 Motivation

While it is clear that previously employed reconstruction methods all have their disadvantages, the necessity of computationally efficient algorithms has never been a topic of discussion. New research in many fields is suggesting a need for a more accurate and computationally efficient method of 3D reconstruction. Their use ranges anywhere from human biomechanics modeling to animating objects in video games. Triangulation-based methods are generally fast enough to represent the models, but the estimations are too vast and the resulting models are not accurate. Volumetric scene recovery methods have generally fixed the quality issue, but are hampered by slow computational speed. As such, it is vital to provide a much more efficient algorithm for these reconstructions.

The art of creating three-dimensional models from two-dimensional images is a fairly new technology that still has many uses yet to be discovered. Through the use of voxel division, it is my hope that the practicality of volumetric scene recovery is greatly enhanced. With this, current and future researchers will be granted a more efficient platform for object reconstruction, hopefully enabling new applications that would have not been possible before.

1.3 Literature Review

Research on volumetric scene recovery is best summarized in the following two surveys: "Volumetric Scene Reconstruction from Multiple Views" by C. Dyer, and "A Survey of Methods for Volumetric Scene Reconstruction from Photographs" by Slabaugh, Culbertson, Malzbender, and Schafer [9,28]. These papers identify the problem of three-dimensional model reconstruction and how it is performed using volumetric scene recovery. Both references discuss many approaches that primarily focused on quality optimization rather than computational efficiency. As such, we decided that our proposed algorithm was prospective enough to conduct the research.

Particularly, these two survey papers provide comprehensive studies in comparing the multiple different techniques. Some examples of volumetric techniques outlined between the two papers are "shape from silhouettes", "shape from photo-consistency", "voxel visibility using plane-sweep", "voxel coloring", and "volumetric pair-wise feature matching" [9, 28]. Each of these techniques approaches volumetric recovery differently with hopes of achieving the highest quality results.

In this research, we chose the method that would best display the capabilities of the voxel division algorithm and could most easily be manipulated to perform a quality reconstruction within the time constraints of this research. The technique we chose is known as the "photo-consistency" method. This method evaluates a voxel's makeup based on the comparison of its projections onto multiple stereo images, detailed further in the following chapters.

Aside from the surveys introduced above, another important paper providing a competitive analysis of recent approaches is “A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms” [27]. Published at CVPR-2006, this paper was written by the creators of the Middlebury College multi-view dataset used in my research. This paper, along with the Middlebury College website, discusses and displays results of many volumetric methods used to reconstruct the given datasets. This information was also valuable in determining which volumetric to use for my research and for providing comparison tools for my eventual results.

1.4 Research Objective

In performing this research project, my goal is to investigate a new algorithm to more efficiently accomplish an established process. While the inputs and outputs would remain the same, the steps in between differentiate my research. Whether or not the voxel division algorithm would be more efficient than other known methods remained as my hypothesis. As such, my objective is to create a functional piece of software, using a new algorithm, which would produce a similar outcome as other established methods with an increase in computational speed.

CHAPTER 2 – BACKGROUND

2.1 Triangulation-Based Method

The most common form of three-dimensional object reconstruction is done using the traditional triangulation-based method [2]. This method back-projects matched feature points across multiple stereo images to 3D object space. The triangulated intersection of these back-projection lines represents the point in three-dimensional space corresponding with the matched feature points across respective stereo images. Using color values from matched pixels on the images, a surface color can be projected to the three-dimensional object. Once all matched feature points have been back-projected to the object space, a point cloud is created representing the surface of the object being reconstructed. The sparsity of matched feature points will directly correlate with the sparsity of the point cloud. A surface can then be interpolated to fit the point cloud and represent the reconstructed object.

Numerous methods have been employed to accurately extract and match like feature points across multiple stereo images. While the point-extraction problem has suitably been solved with algorithms such as *histogram-oriented gradient* (HOG) and *scale-invariant feature transform* (SIFT), feature point matching still remains an issue [12,21]. The most accurate method for matching alike features is by manual selection. This method is most accurate because humans have the ability to easily comprehend

correspondences in like features more so than a computer. However, this method is also the most time-consuming. As such, if one wants to acquire a dense set of matched feature points without employing a large amount of time, a computerized method must be used.

Common feature point matching algorithms range from mathematically simple to complex [18]. Typically, simple methods are less effective in matching feature points, but easier to understand, construct, and implement. A few common simple feature point matching techniques include sum of squared differences, absolute difference, and normalized correlation. In attempting to reconstruct high-quality three-dimensional objects, it is important to have a dense set of matched feature points. Recently, advanced point-matching algorithms have been generally more successful in matching dense sets of matched feature points. A few of these algorithms include pose clustering, feature focusing, and random sample consensus (RANSAC) [29].

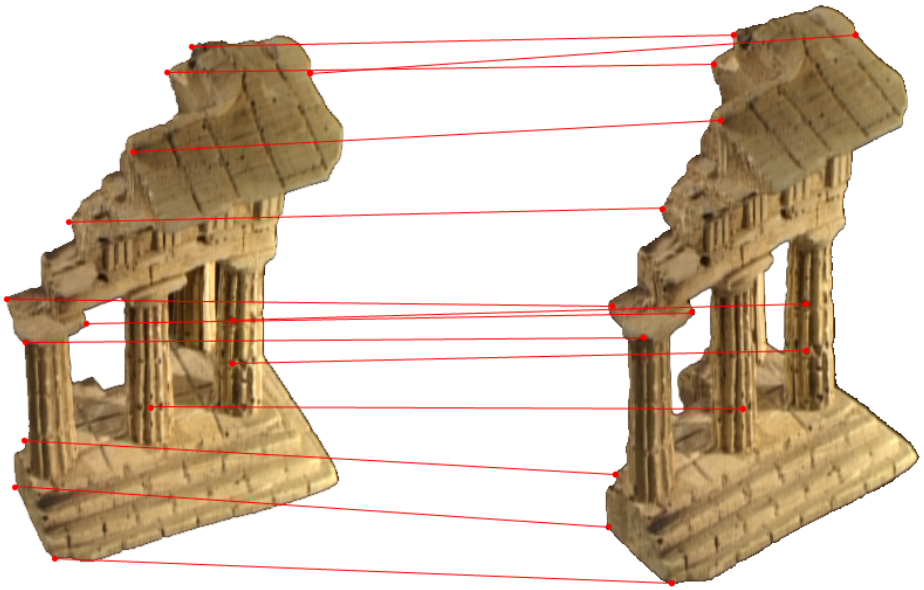

Figure 1 – Example matched feature points across two stereo images illustrated using manual selection [27]:

After a satisfactory amount of matched feature points are obtained across the stereo images, corresponding points are back-projected to the three-dimensional object space. Figure 2 shows the geometry of the multiple camera views and their projections to the surface in the object space.

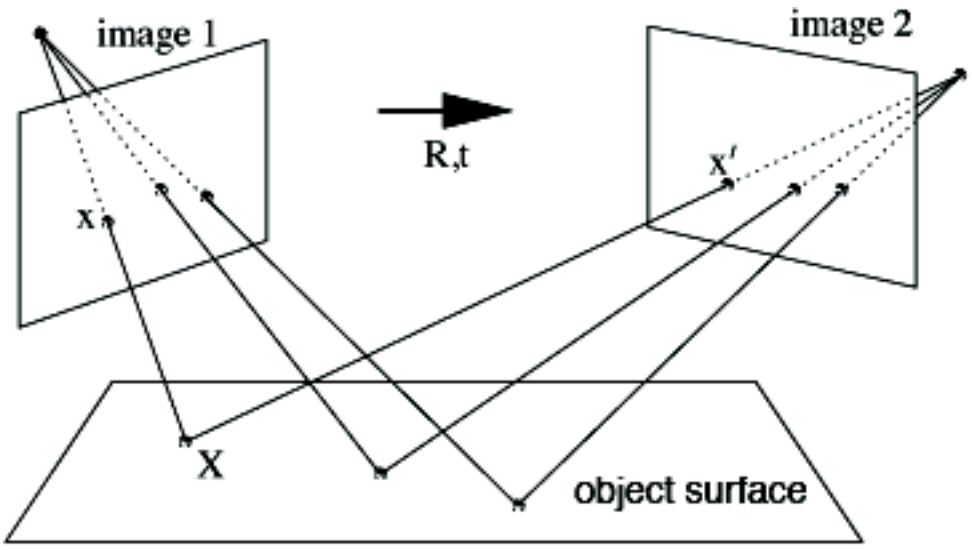

Figure 2 – Geometric orientation of triangulation of matched stereo feature points to three-dimensional object surface [13]:

Once all matched feature points have been projected into the 3D object space, the resulting point cloud will represent the surface of the three-dimensional object being reconstructed. Surface fitting a point cloud is typically done using a polygonal meshing algorithm that connects points along the surface and fills in the resulting polygons [8,22]. Typically, Delaunay triangulation is used on the point cloud with each triangle assumed as a 3D plane. The surface-fitting technique used in this research is called the marching cubes algorithm, explained in detail in Chapter 3. The quality of the completed interpolated surface directly corresponds to the sparsity of matched feature points and their resulting point cloud in 3D object space.

As is, the traditional triangulation-based method for modeling three-dimensional has been the standard for many years. However, recent needs to produce highly accurate three-

dimensional models have pointed out significant flaws in using triangulation to perform reconstruction. One major limitation in triangulation-based methods arises in feature point extraction. Since features (e.g. corners on a building) are determined at an integer level (i.e. point coordinate being in the center of the pixel), correspondences between matched points are not exact. When this issue is added to existing inconsistencies in camera parameters, back-projected lines do not always converge [3]. To adjust for this, a bundle adjustment is necessary, adding a high degree of estimation to the calculations.

Another limitation in triangulation-based methods arises in matching the extracted feature points. Though recent algorithms have become more effective in matching feature points, there are still many instances where doing so is not feasible. Textures, shadows, and repeating patterns are a few of the things that can make image point matching incredibly difficult. Imagine trying to match feature points on a brick wall or in dense foliage where repeating patterns make several areas on the image look identical. No algorithm is intuitive enough to differentiate the bricks or leaves apart. As such, the resulting matched feature points will either be very sparse, incorrect, or nonexistent.

Figure 3 – Repeating patterns, such as on a brick wall or on leaves blowing in the wind, can provide incredible difficulties when trying to match alike features across stereo views [15,16]:

In a similar vein, it is equally as difficult to match feature points on images with a large viewpoint change (i.e. wide camera baseline). If the baseline between images is too wide or the images have inconsistent zoom, the object's representation on the image will skew. Imagine viewing a tall building from an airplane from two separate angles, one looking near-nadir and the other looking from a side angle. The side-looking view will show the tall building as elongated, whereas the other view will skew the building to look very short. Whereas a human can intuitively discern two points on the image as alike, a feature point matching algorithm will typically become confused by the skew, scale, and shadow changes.

Another issue plaguing the triangulation-based method is the necessity for correspondences between camera parameters and matched feature points to be maintained over many views spanning large viewpoints. As the number of stereo views increases, as does the complexity of the system. This issue typically results in time-consuming computations to keep all of the data organized and correctly correlated. This is an

inefficient task, adding unnecessary time to the preprocessing stage of object reconstruction.

All in all, triangulation-based three-dimensional object reconstruction can be effective in some instances. However, in new practical applications requiring the use of three-dimensional renderings the sparsity, large amounts of estimation, and susceptibility for error is too great. As such, the need for a new algorithm with the ability to provide higher quality models has led to recent research and implementation of volumetric scene recovery.

2.2 Traditional Volumetric Scene Recovery

Volumetric scene recovery works opposite to that of traditional triangulation-based object reconstruction. As opposed to triangulating from image space to object space, this approach performs calculations in three-dimensional space by projecting a voxel's bounding vertices to their corresponding regions on all stereo images [9,28]. To perform a high-quality reconstruction, a highly granular grid of voxels is considered in the object space. Each voxel is represented as a cuboid with six sides whose vertices have the bounds [(*Xmin*,*Ymin*,*Zmin*),(*Xmax*,*Ymax*,*Zmax*)]. These eight vertices, when projected to the stereo images, create polygons that contain the 2D representation of the voxel from each camera's viewpoint. By comparing the histogram of pixels contained in the polygon bounds on one stereo image to other images at different viewpoints, a general representation for the voxel can be formulated. In advanced forms of volumetric scene recovery, three histograms (red, green, and blue) are analyzed together to determine the

voxel color. A simpler version, as researched in this thesis, analyzes grayscale images and labels voxels as either transparent or opaque. Once all voxels defined on the three-dimensional grid are analyzed, those that are labeled as opaque make up the reconstructed object.

In order to properly execute a volumetric scene recovery, three variables are needed: camera parameters, stereo images, and transformation equations. Camera parameters are given for each image in the dataset as *K*, *R*, and *t* [2]. *K*, as shown below, is a 3 x 3 matrix containing the intrinsic camera parameters.

$$\mathrm{K} = \begin{bmatrix} a_x & s & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The variables a_x and a_y refer to the focal length, in pixels, along the x and y directions; x_0 and y_0 represent the principal point; and *s* represents the shearing. *R* is a 3 x 3 matrix and represents the camera rotation and *t* is a 3 x 1 matrix representing the camera translation.

With these known camera parameters, the projection matrix *P* (3 x 4) can be modeled for each image using the following equation:

$$\mathrm{P} = \mathrm{K}[\mathrm{R}|\mathbf{t}]$$

$$\mathrm{P} = \begin{bmatrix} a_x & s & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_{11} \\ R_{21} & R_{22} & R_{23} & t_{21} \\ R_{31} & R_{32} & R_{33} & t_{31} \end{bmatrix}$$

With the projection matrix calculated for each image, the eight vertices of the voxel can be converted to two-dimensional polygon boundary points for each stereo image. Using

the three-dimensional vertex points matrix *B* (4 x 8), the two-dimensional points matrix *Q* (3 x 8) can be computed as

$$\mathrm{Q} = \mathrm{P}^{*}\mathrm{B}$$

$$\begin{bmatrix} x_1 & x_2 & . & . & x_8 \\ y_1 & y_2 & . & . & y_8 \\ k_1 & k_2 & . & . & k_8 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} * \begin{bmatrix} x_1 & x_2 & . & . & x_8 \\ y_1 & y_2 & . & . & y_8 \\ z_1 & z_2 & . & . & z_8 \\ 1 & 1 & . & . & 1 \end{bmatrix}$$

The resulting matrix *Q* contains unscaled values for *x* and *y*. The values for *k* represent the scale factors for each coordinate set, so the final scaled values of *x* and *y* can be calculated as:

$$\begin{bmatrix} x_1/k_1 & x_2/k_2 & . & . & x_8/k_8 \\ y_1/k_1 & y_2/k_2 & . & . & y_8/k_8 \end{bmatrix}$$

With the vertices projected onto each stereo image, a convex hull algorithm can be used to determine the polygonal boundaries. This process, discussed in more detail in Chapter 3, connects the dots to form boundaries, thus turning the scattered points into a polygon. Pixels contained within this polygon are the projection to the respective stereo image of the current voxel being analyzed. In many cases, the line of sight from the camera to the voxel is obstructed by some other part of the object. As such, note that the pixels within the polygons on the stereo images are only where the voxel projects to, and not necessarily exact visual representations of the voxel. The entire projection can be visualized geometrically in Figure 4.

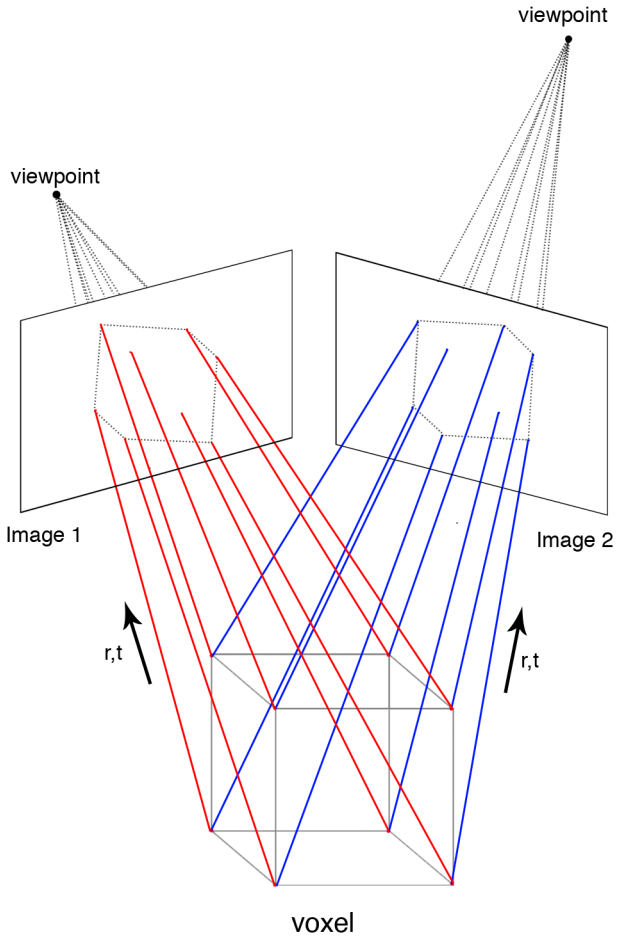

Figure 4 – Visual representation of geometry used to project from 3D voxel to 2D stereo images:

Next, histograms are computed for the area within the polygon on each stereo image. In the volumetric method being used, a voxel can be determined transparent or opaque based on comparing histogram background-foreground color ratios to each other. If any histogram from an image is predominantly colored the same as the background color (black in our case), it can be devised that the voxel does not lie on the object surface and is thus transparent. Likewise, if every histogram is predominantly colored the same as the

object's color (gray in our case), the voxel must be part of the object. In this case, the voxel is labeled opaque and added to the point cloud representing the surface of the object.

In traditional volumetric scene recovery, the process of 3D to 2D projection and histogram comparison is repeated for every voxel along the dense grid in the object space [9,28]. This grid consists of a highly granular set of voxels that combined form a single cuboid surrounding the object being reconstructed. With every voxel analyzed, each subsection of the three-dimensional object space becomes classified as transparent or opaque. Together, the voxels labeled as opaque will form the point cloud for the three-dimensional model. From this point cloud, surface fitting can be completed in the same fashion as in using the triangulation-based method.

As can be noted, the quality of the resulting point cloud is directly correlated to the granularity of the grid of voxels. For example, if the grid contained only 100 voxels, the histogram comparison would not be able to accurately determine if the voxel was transparent or opaque. This is rightfully so, since a voxel of that size most likely contains both transparent and opaque regions. However, the benefit of volumetric scene recovery is that one can include millions of voxels in their grid in order to achieve great accuracy. Unfortunately, a higher granularity of voxels results in a higher number of computations. Hence, the computational complexity of performing high accuracy volumetric scene recovery can be incredibly high. Even if a single voxel computation takes mere

milliseconds, the accumulation of millions of voxel computations can put processing time into a number of days or weeks.

Having described both methods, it is apparent that triangulation-based and volumetric-based reconstructions each have their own setbacks, whether it is model quality or computational complexity. With the growing need for a highly accurate, yet near real-time method for modeling three-dimensional objects, it is imperative to improve the speed of volumetric scene recovery. The voxel division algorithm aims to do just that.

2.3 Voxel Division Algorithm

The voxel division algorithm can be used in conjunction with the same basic principles of any form of volumetric scene recovery. Voxels in three-dimensional object space are represented as cuboids whose vertices are then projected to the corresponding points across all stereo images. By comparing histograms of projected regions, it can be determined if a voxel is opaque or transparent. Resulting voxel point clouds are used to perform surface fitting to reconstruct the object.

What sets the voxel division algorithm apart from traditional volumetric methods is the way the voxel grid in the object space is formed. Using the voxel division approach, the first computation is performed on one voxel. The bounds of this voxel are the same as those of the object in that the voxel creates a tight-fitting box around the object. As is the case with traditional volumetric scene recovery methods, the voxel is then projected to the stereo images. Once histograms have been generated for all of the polygons on the

stereo images, the background-to-foreground color ratio is analyzed. In the first iteration, the polygon will surround the entire object on the stereo images, so the histograms will be split between foreground and background colors. In this case, it is obvious that the voxel cannot yet be classified as opaque or transparent.

The inability of the histogram comparison to classify the voxel as transparent or opaque stress that the grid must be more granular (i.e. more voxels) to achieve a higher quality result. To do this, the current voxel being analyzed is split into eight subvoxels with the analysis process restarting again for each new sub-voxel. Each time a new voxel is analyzed, it must meet one of four conditions. The first two conditions are fulfilled respectively if the voxel is classified as opaque or transparent. In this case, no more subdivision of the voxel is necessary as it is understood that any subvoxels within this voxel will retain the same classification.

To ensure that voxels do not subdivide forever (e.g. down to a single pixel level), the third condition sets a division limit. The limit itself is an arbitrary figure and has been tested at a 100, 10, and 3-pixel level for this research experiment. This number corresponds to the average number of pixels contained within the projected polygons across all stereo images. A lower limit (e.g. 3-pixel) allows voxels to subdivide further and achieve a higher quality, but at an increase in processing time. If a voxel reaches this limit, the predominant color (foreground or background) at that point will determine opacity or transparency.

The final condition is fulfilled if a voxel cannot be deemed transparent or opaque and is also still too large to estimate. This is the case in which the voxel will divide into eight subvoxels. Within the first divisions of the bounding voxel, this will likely be the predominant condition to be fulfilled. This process will continue to subdivide recursively until voxels meet one of the first three conditions.

When all recursions are finished, a final matrix of opaque and transparent points is produced as in traditional volumetric scene recovery. Unlike the former method however, not all voxels cover the same volume in three-dimensional object space. The point of the voxel division method is to eliminate redundancy created in traditional volumetric scene recovery techniques. For example, consider an object that is 'L' shaped. A bounding box around this object would contain a large portion of empty space that would be analyzed in different fashions in the two methods. Whereas typical volumetric scene recovery techniques would analyze every single transparent granular voxel, the voxel division technique would generalize the entire area as transparent with no need for any more subdivision. My experiments determined that my test object required seven layers of recursion (i.e. 8^6 total granular voxels) to create a reconstruction model at 10-pixel accuracy. To reconstruct the object at the same accuracy, traditional volumetric scene recovery would calculate all 8^6 voxels independently. The experiment using voxel division did not need nearly as many calculations, since the majority of voxels were generalized as opaque or transparent before ever reaching such a granular level.

In performing many experiments using the voxel division algorithm, the magnitude of redundancy and unnecessary calculations performed by traditional volumetric scene recovery methods became increasingly evident. A more technical report of the methodology and implementation of the algorithm is described in the following chapter.

CHAPTER 3 – METHODOLOGY

3.1 Dataset

The datasets used in my research were provided by the Middlebury College Stereo Vision Department [24]. These datasets are publicly available for use as a research project at Middlebury College concerning the comparison and evaluation of multi-view stereo reconstruction algorithms [27]. The dataset's website states "the goal of this project is to provide high quality datasets with which to benchmark and evaluate performance of multi-view stereo reconstruction algorithms. Each dataset is registered with a ground-truth 3D model acquired via a laser scanning process, to be used as a baseline for measuring accuracy and completeness [24]."

There are two main multi-view datasets provided by Middlebury College, each containing two sampled sub-datasets. The main dataset used for my reconstruction work is the "Temple" dataset. The object captured is a "plaster reproduction of the Temple of the Dioskouroi in Agrigento, Sicily [24]." This dataset contains a large amount of fine detail as well as background-foreground breaks, making it a great test for both the accuracy and efficiency of my algorithm.

The second dataset provided is the “Dino” dataset consisting of stereo images of a simple plaster Dinosaur. This dataset, as opposed to the Temple dataset, has a less reflective surface, thus increasing shadows. This dataset was useful in comparing the algorithm’s accuracy in shadowy areas, but lacked enough texture and foreground-background color changes to prove as a good visualization tool for the reconstruction. Due to this observation, the majority of the quality tests and figures in this thesis focus on the Temple datasets.

Figure 5 – Example dataset images showing the Temple and Dino models provided by the Middlebury College Stereo Vision Department [27]:

Each of the datasets was captured using the Stanford Spherical Gantry, a large rotating arm on which cameras can be mounted for 360° stereo coverage [24]. Camera calibration

parameters were collected as the images were captured so that all resulting data would be synchronized. The full stereo coverage of the Temple dataset contains 312 views, whereas the Dino dataset contains 363. Within these, each dataset has two sampled datasets: “Ring” and “SparseRing”. The TempleRing and DinoRing datasets each contain 48 sampled views, while the TempleSparseRing and DinoSparseRing datasets contain a smaller sampling of 16 views. Each sub-dataset contains full stereo coverage of the object, though the larger sets increase quality via redundancy in histogram analyses. As usual, the larger datasets also take an increased amount of processing time. The differences in time and quality between these sub-datasets can be seen in the Visual Results section of Chapter 4. Figure 6 shows the hemisphere over which the camera shots were taken. The red triangles represent the SparseRing dataset views, while the red and blue triangle together represent the Ring dataset views. The full dataset contains views from all triangles; black, blue, and red.

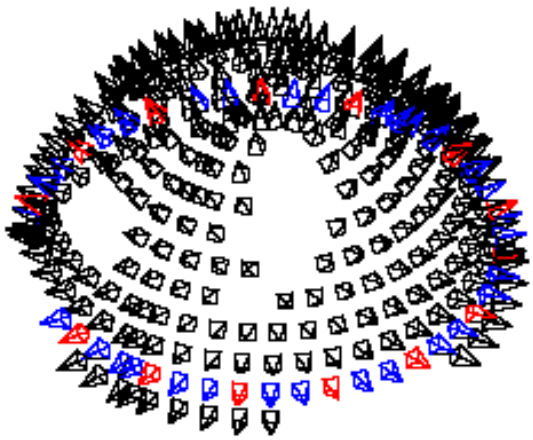

Figure 6 – Full, Ring, and SparseRing dataset viewpoints visualized on the hemisphere created by the Stanford Spherical Gantry [27]:

In order to judge the quality of the reconstructed model, we compare results to the closest representation of the object that is available. Fortunately, a ground truth model of the datasets is provided on the Middlebury College website. Completed models created by other researchers are also available and provide a precise tool for method comparison. Figure 7 shows the ground truth model of the Temple, the primary dataset used for my experiments.

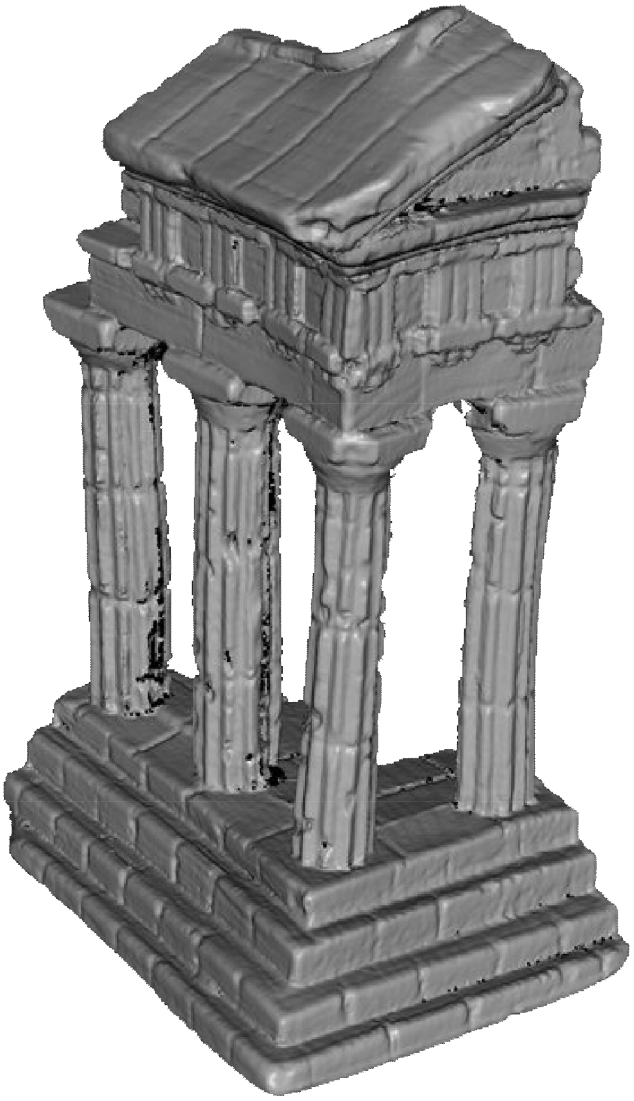

Figure 7 – Ground truth model of the Temple dataset [27]:

3.2 Preprocessing

Before implementing the proposed algorithm, a small amount of dataset preprocessing was needed. I first converted all of the images in the datasets from RGB to grayscale. The scope of my research is to determine if using a voxel division technique will enhance the speed of volumetric scene recovery methods, so using RGB histograms is not going to affect computational comparison. Due to this, all of my calculations were done in grayscale using one-dimensional histograms. This allowed me to focus on speed and voxel classification rather than measuring consistency in voxel colors.

With the images converted, it is then necessary to import the given camera and object scene parameters. The camera orientation parameters K, R, and t are provided for every image in each dataset in a large text file. Using built-in MATLAB functions, the text was imported and transformed into matrices for easier processing. Also provided with the dataset was the tight bounding box for the three-dimensional object. This was provided in the form [($Xmin$,$Ymin$,$Zmin$),($Xmax$,$Ymax$,$Zmax$)], defining the first voxel for which to do the recursive analysis. It is not absolutely necessary that the tight bounding box be given to perform volumetric scene recovery, but its addition is helpful in reducing errors. Without it, a best-guess estimate would be necessary for the original voxel size in three-dimensional space. This could provide computational difficulties and result in missed areas (not enclosing enough object space) or excess calculations (enclosing too much object space).

With the camera parameter matrices and images imported, the *P* projection matrix is calculated for all images. Since the stereo images and their respective projection matrices are used continuously throughout the recursive analysis, it is imperative to eliminate redundant importations and calculations. As such, both matrices are imported and calculated only once at the beginning of the algorithm and stored in a cellular fashion as to not slow processing time. In the final step of the preprocessing stage, other variables are determined for later use such as the image size (480 x 640 in our datasets) and total number of stereo images in the dataset being used. These are very important variables later during the analysis when projecting points and creating polygon masks. With all vital input variables defined, the recursive projection and analysis stage of the algorithm begins.

3.3 Voxel Projection

As given in the dataset, the vertices of the tight bounding box around the object define the first voxel to be analyzed. The voxel's bounding box is given as minimum and maximum bounds in the object space from which the eight corners of the box can be computed. These eight corner vertices can be connected to provide a visual representation of the voxel in the object space, as seen in Figure 8.

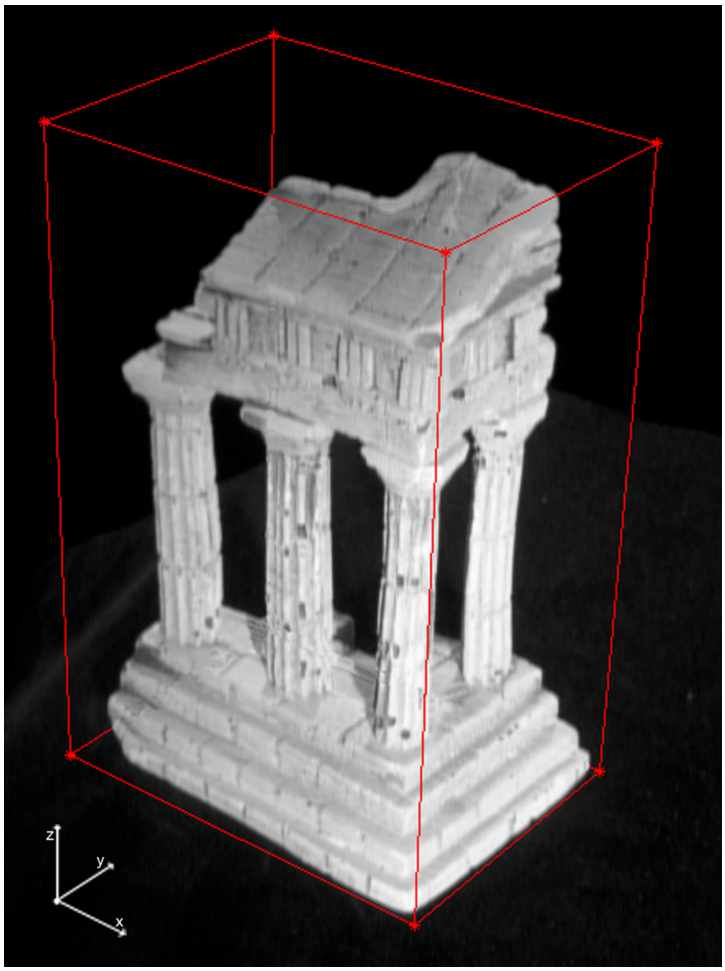

Figure 8 – The Temple dataset model is seen in 3D space with the first voxel overlaid, represented by a bounding-box formed from the eight vertices [27]:

The voxel vertices are next projected from the object space to each of the stereo images. This projection is performed by multiplying the projection matrix *P* with the voxel vertices matrix *B,* as explained in Chapter 2. The resulting matrix *Q*, after applying the scale factor *k*, contains the two-dimensional representations of the three-dimensional points [2]. These coordinates can be visualized on any of the stereo images, as seen in Figure 9.

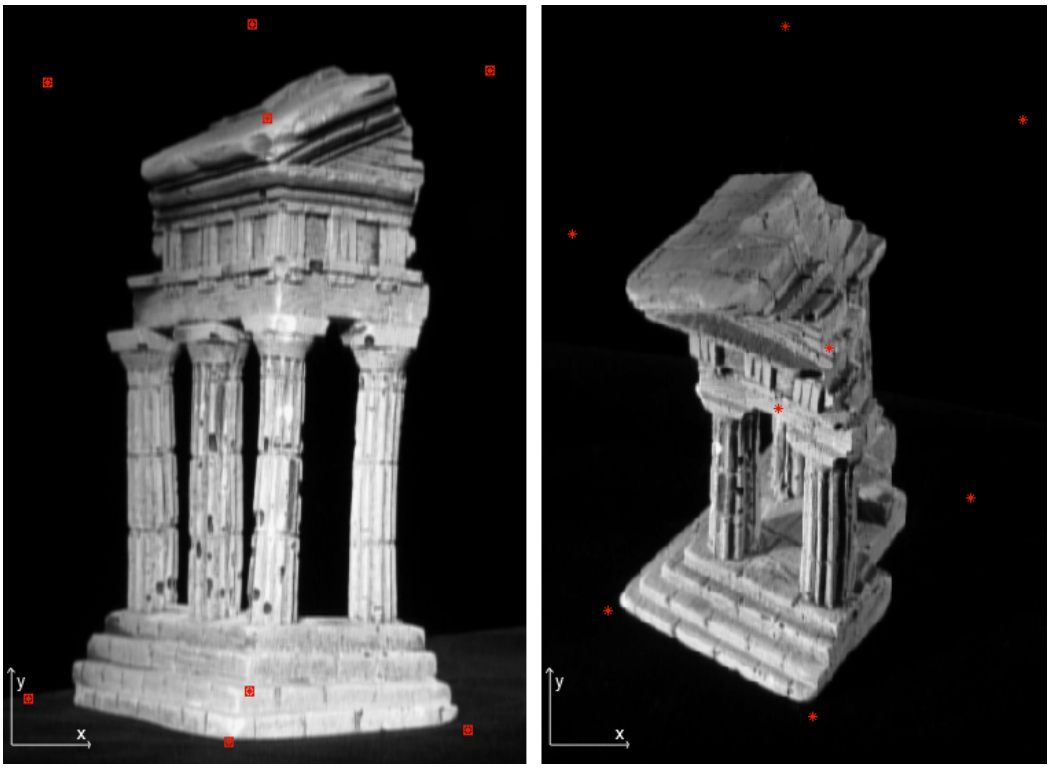

Figure 9 – Images 12 (left) and 134 (right) from the Temple dataset are overlaid with points projected from the three-dimensional voxel vertices [24]:

With the eight points projected into two-dimensional space, it is necessary to determine which pixels are needed for the histogram analysis. To do this, the external boundary created by the points on each image must be computed. As seen in the two images above, two of the points are contained within the boundary created by the external points and can thus be ignored.

In order to determine the points making up the external boundary and the order in which they are traversed, a convex hull is computed [6]. A convex hull is described in simple

terms using the “elastic band analogy”, wherein if an elastic band were stretched to encompass all points, when released it would take the form of the convex hull.

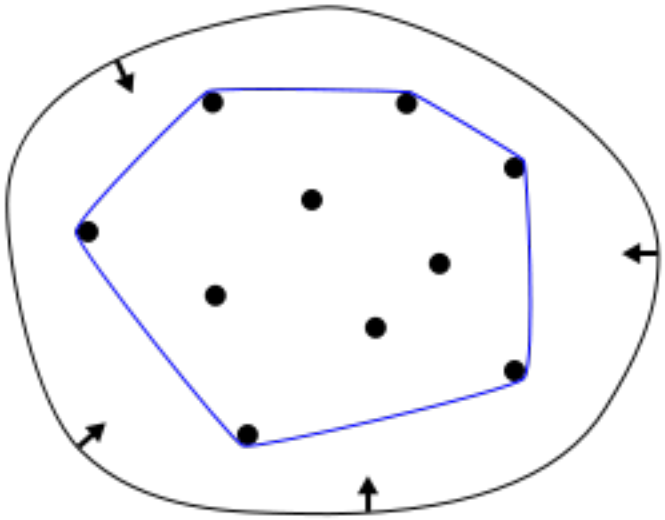

Figure 10 – Convex hull elastic band analogy. The black line represents a stretched band, whereas the blue line represents a released band [14]:

There are many methods for determining a convex hull, all of which are mathematically advanced. Fortunately, MATLAB provides the built-in function *convhulln* that computes the convex hull of a set of points using the QuickHull algorithm [7]. The QuickHull algorithm, discovered independently by two different researchers in 1977 and 1978, is a faster replica of the earlier QuickSort algorithm [5,10]. These algorithms use divide-and-conquer strategies to sort through all subsets of points given. For each subset analyzed, calculations are performed to determine which point is the most external in the scene. Eventually, all subsets are sorted and the convex hull is created. Figure 11 shows a visual representation of the projected points after being sorted using a convex hull. Lines are

drawn between points to show the polygonal boundary containing the pixels to be analyzed.

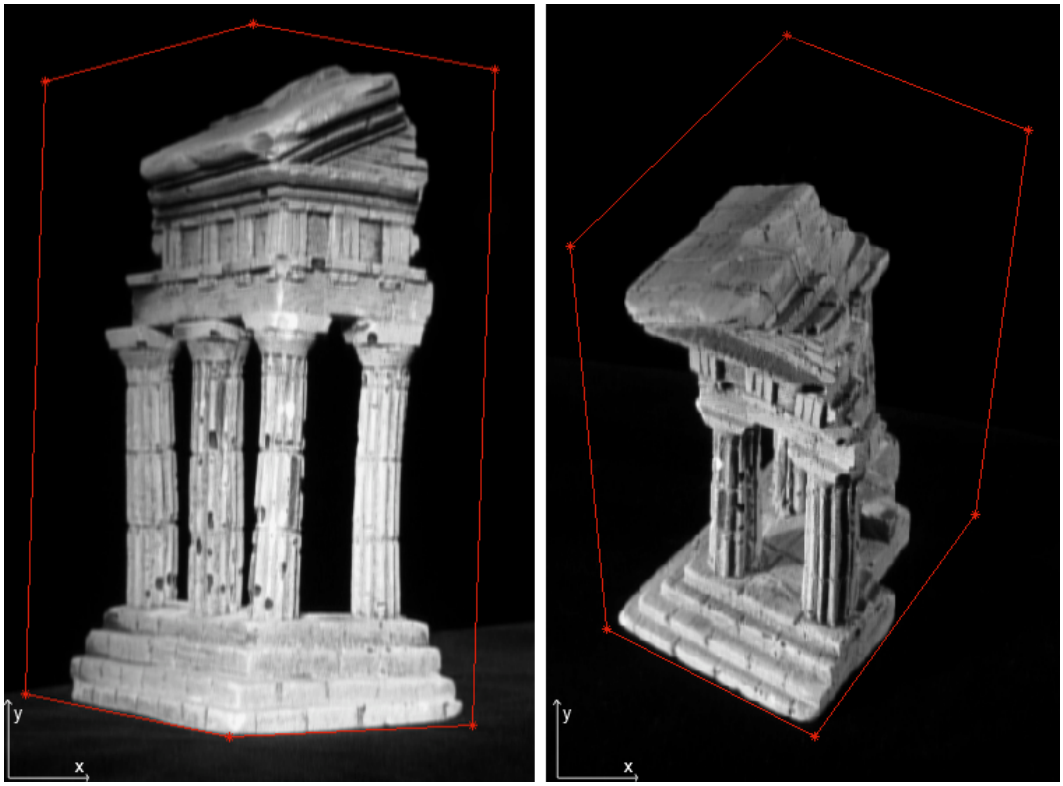

Figure 11 - Images 12 (left) and 134 (right) from the Temple dataset are shown after having the projected points sorted using the convex hull algorithm. Interconnected lines define the boundaries containing the pixels to be analyzed [27]:

With redundant projected points eliminated and boundary lines drawn, the next step is to mathematically determine the pixels contained within the boundaries on each stereo image. To do this, a polygon mask is created. As with the convex hull algorithm, MATLAB provides a built-in function called *poly2mask* to perform this task. Given the ordered set of six points defining the boundary of the polygon, the *poly2mask* functions outputs a binary mask matrix for the entire image [26]. In our case, the resulting mask

matrix retains the same dimensions as our image. In this matrix, pixels whose center points fall within the boundary polygon are labeled with a *1*, while all pixels whose center points remain outside the boundary are labeled with a *0*. Assuming *0*'s as black and *1*'s as white, the resulting mask matrix can be visualized as seen in Figure 12.

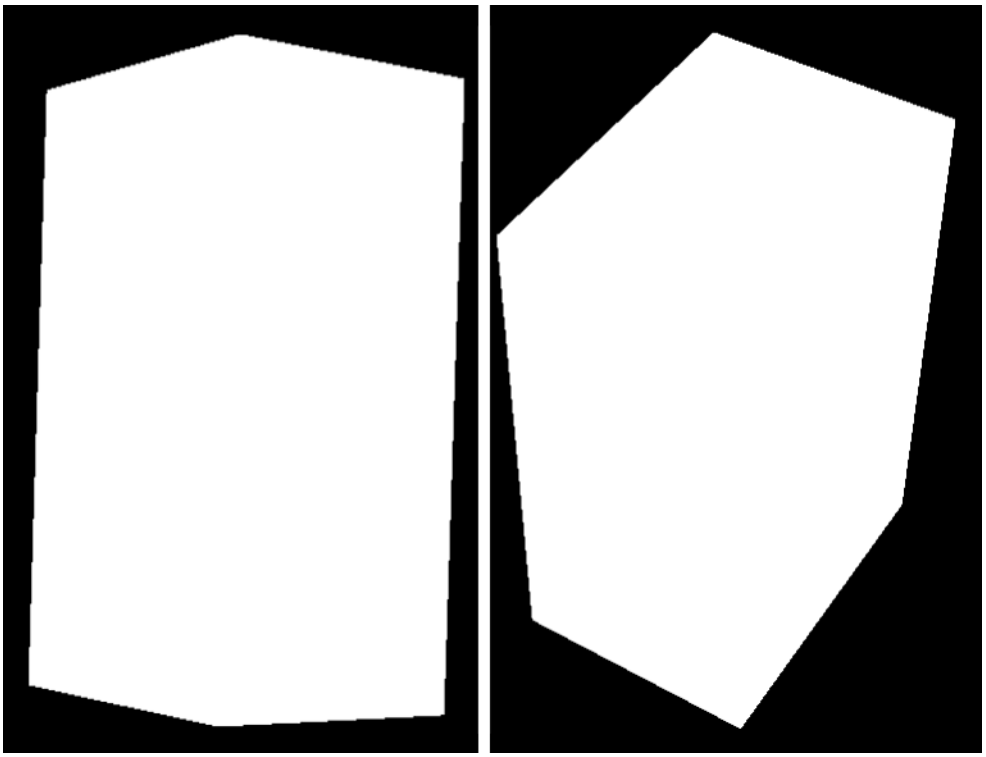

Figure 12 – Images 12 (left) and 134 (right) overlaid with their respective boundary-defined polygon masks. White pixels are those contained within the boundary, whereas black pixels are those outside:

With the polygon mask matrices defined, the MATLAB built-in function *find* can be used [11]. This function, given a matrix and a search variable, will return all indices in said matrix where the variable appears. Based on the masked pixels on each of the stereo

images, histograms are generated and compared to determine whether or not the object is contained inside the projected voxel.

3.4 Histogram Analysis & Voxel Division

With the pixels contained within the polygon boundaries determined, a 64-bin histogram is created for every image. Since all images are analyzed in grayscale, a single histogram suffices during comparison. As opposed to comparing histograms across all images for color similarities with RGB images, grayscale images enable a simpler test of opaqueness or transparency. In our dataset, the background color is near black, whereas the foreground color (the object) contains shades of light gray. The colors change depending on dataset and need to be updated inside the algorithm if a different dataset in employed.

Using the generated histograms, a *background-color to foreground-color* ratio is determined for every image. The threshold on the histogram where values change between foreground and background color, or in our case where the values are no longer considered black, is arbitrary and changes with the dataset. In the case of the Middlebury College datasets, a large spike of background color (black) appears in the range from bins 0 to 3. As such, a sum of the histogram values between bins 1 and 3 was compared to those between bins 4 and 64 to create the background-foreground ratio. As a result of the entire histogram being subdivided into two generalized sections, a high bin count (e.g. 255) is not necessary. 64 bins were used in the proposed algorithm, though a smaller number of bins would likely increase speed even more with similar results.

Having determined ratios for all histograms, the voxel is analyzed to determine which of the four conditions it falls under. The first condition applies if *any* histogram across all stereo images contains more than 90% background color. In this case, at least one camera viewpoint is seeing only background color when projecting to the current voxel, thus showing that it is transparent. In this case, vertex coordinates of the current voxel are added to the final matrix and labeled transparent. No more analysis or division is needed for this area, so the algorithm moves on to other voxels. In Figure 13, a sample histogram shows a case where the voxel is determined to be transparent.

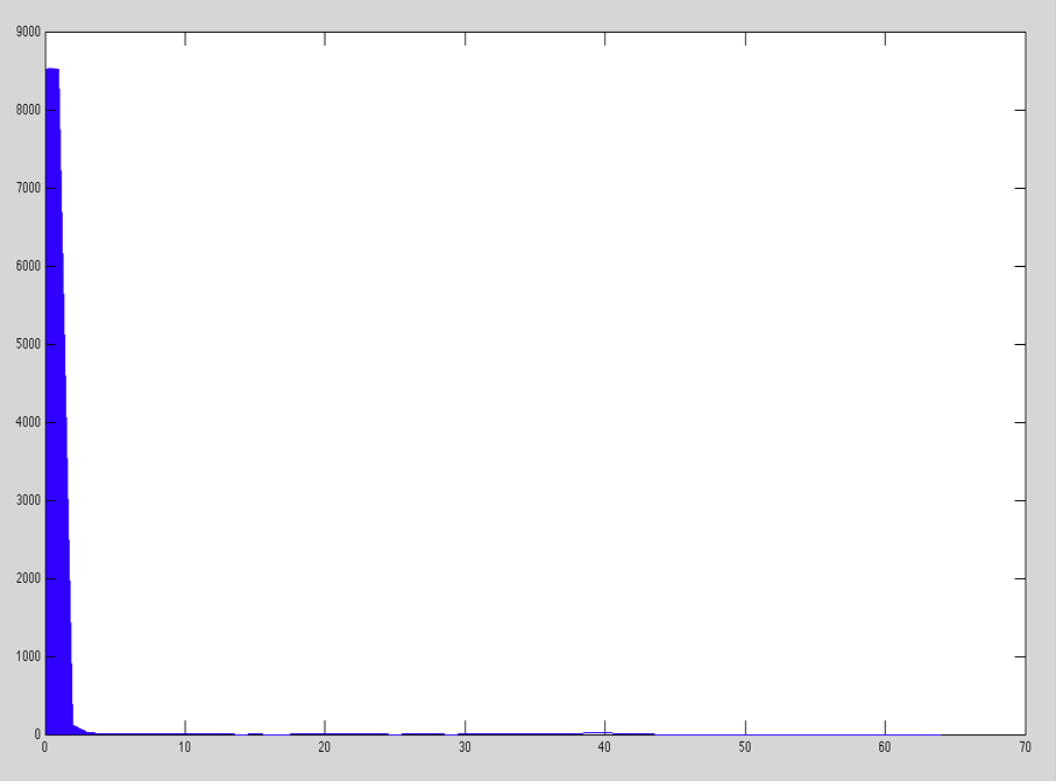

Figure 13 – Sample histogram showing a case in which condition one would be satisfied. This histogram contains ~98% background color:

If the histograms do not meet the requirement of the first condition, the second condition is considered. Condition two is the opposite of condition one, stating that if *every* histogram has less than 10% background color, the voxel is opaque and lies on the object surface. Assuming that the stereo images give sufficient 360° horizontal and 180° vertical coverage, a voxel can be classified opaque if every camera sees the projected voxel as foreground color. As with condition one, no more analysis or division is necessary if condition two is met. The voxel can be submitted to the final matrix and labeled as opaque and the algorithm can move on to another voxel. In Figure 14, a sample histogram shows a case where a voxel is determined to be opaque.

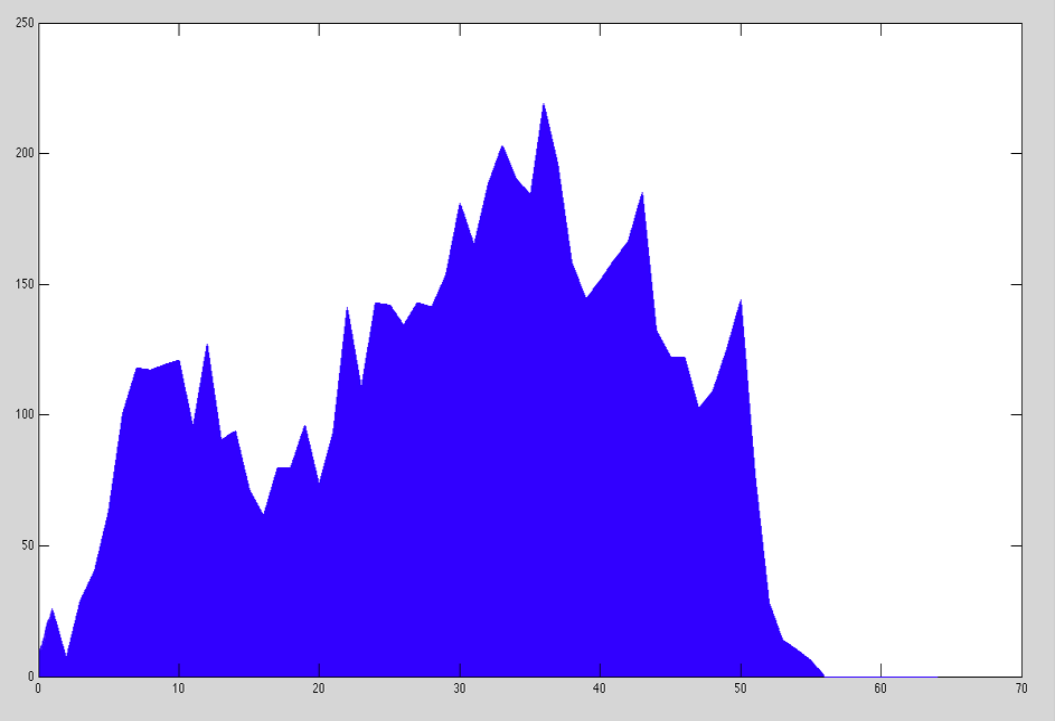

Figure 14 – Sample histogram showing a case in which condition two would be satisfied. This histogram contains ~2% background color:

It should be noted that the 10% and 90% ratios are arbitrary and will affect quality and processing time when changed. A larger upper limit or smaller lower limit will require further division of the voxels to separate background and foreground colors more, smoothing rough edges in areas of contrast between foreground and background colors. I found that for my particular datasets, these values provided an acceptable balance between quality and speed.

If the histogram analysis has proceeded through the first two conditions, there is not yet enough separation between foreground color and background color in the histograms of the stereo images to classify the voxel. In this case, the next step would be to subdivide the voxel. However, a third condition is implemented before reaching that step. It is possible in some cases that a voxel will subdivide many times and still retain subvoxels whose color cannot be determined. In this case, if the algorithm were allowed to subdivide without restriction, the projected polygon boundary would eventually contain zero pixels, thus breaking the code with an empty histogram. To ensure that this does not happen, the third condition contains an escape clause. An arbitrary number of pixels, A, is selected such that if the projected polygon contains fewer pixels than A, the voxel will be estimated based on majority color and will finish subdividing. In empirical tests, I compared 3, 10, and 100 pixel thresholds and determined that a lower threshold resulted in more calculations and longer processing time, but better quality. The visual results can be seen in Chapter 4.

For voxels large enough as to not satisfy condition three whose makeup is between 10% and 90% background color, the fourth condition is necessary. Forming the basis for my algorithm, the fourth condition implements the voxel division technique. By dividing a voxel into eight subvoxels, foreground and background colors are further separated allowing voxel classification to more precisely be determined. In the first iteration of the algorithm, the first voxel encompasses the entire object; hence it is obvious that condition four will be reached. Figure 15 shows an averaged histogram of the first voxel.

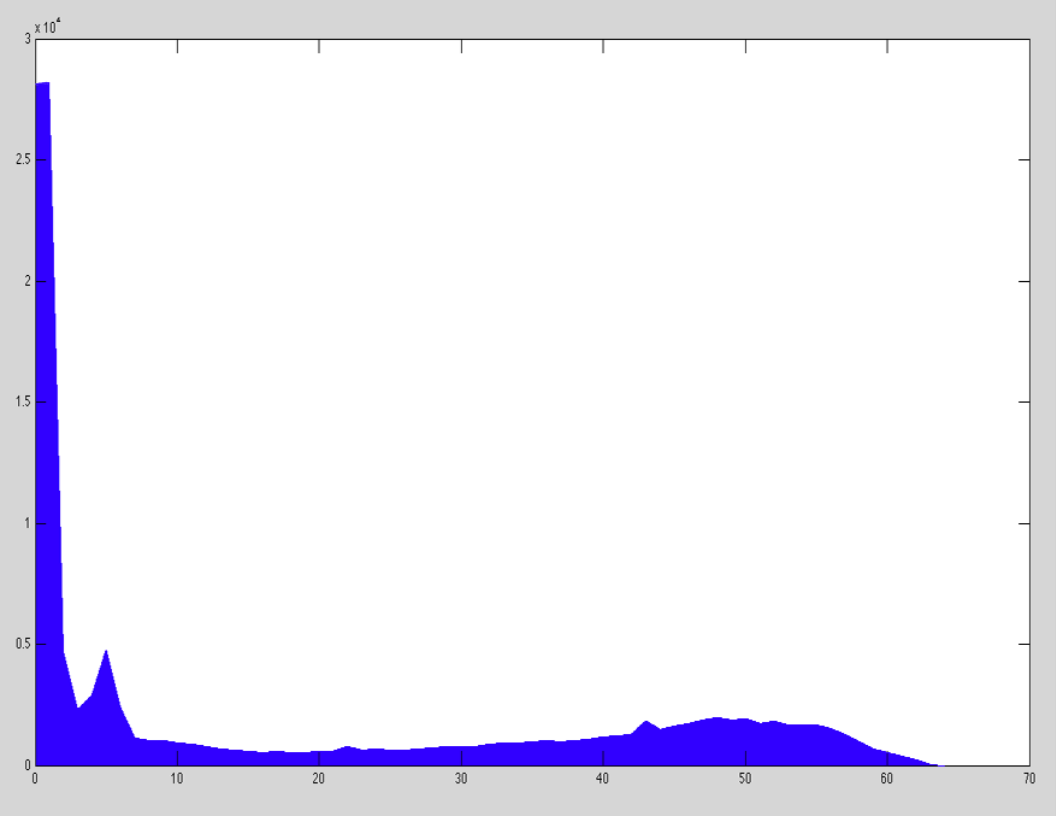

Figure 15 – Sample histogram showing a case in which condition four would be satisfied. This histogram contains ~34% background color:

Voxel division is performed simply by calculating the eight corner vertices for each of the new eight subvoxels based off of the original vertices on the parent voxel. Figure 16 shows the original bounding box voxel after being divided into eight subvoxels.

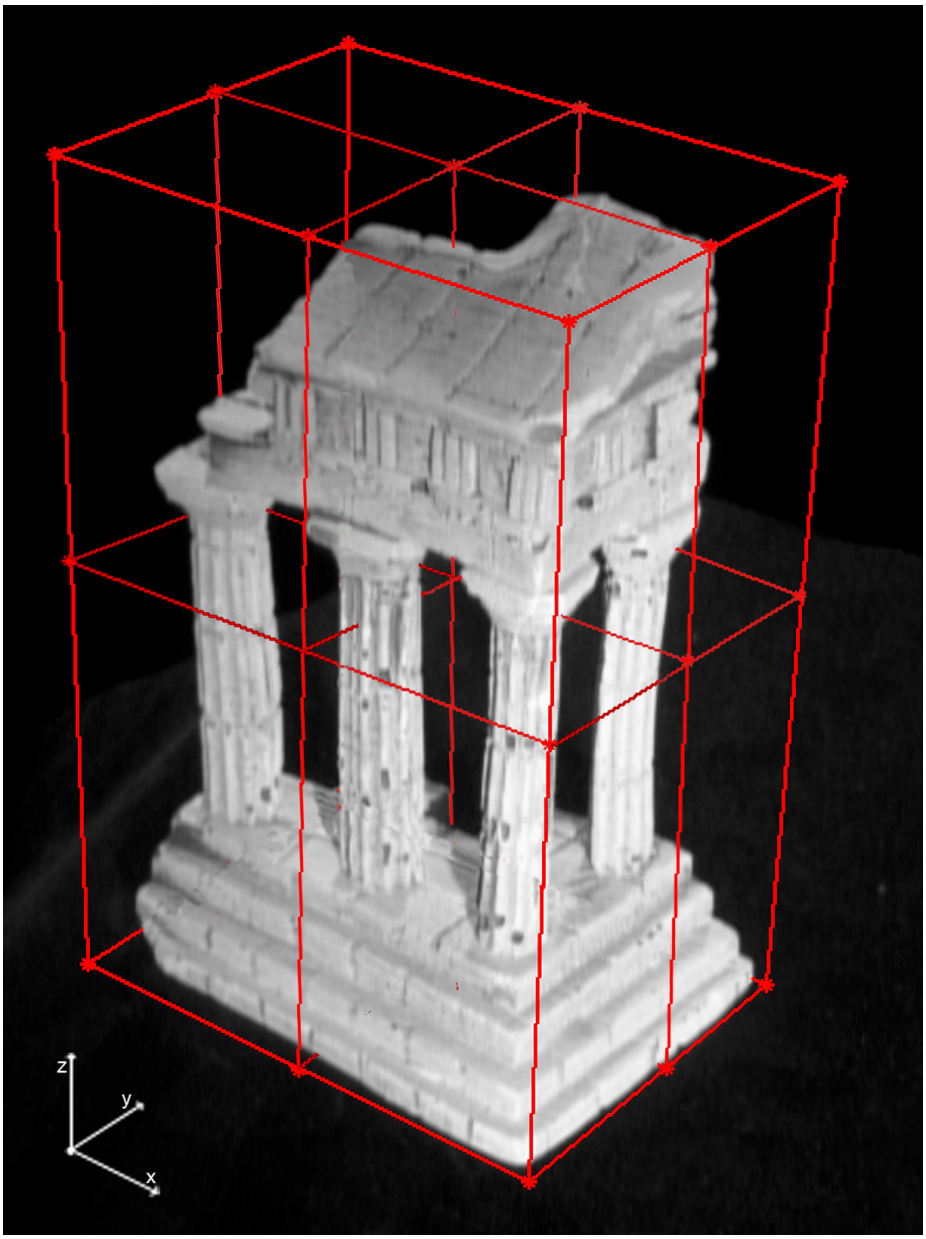

Figure 16 – The original bounding box voxel is subdivided into eight equal volume sub-voxels [27]:

To perform the recursive aspect of the division, the entire process of projection, convex hull, polygon mask, histogram analysis, and conditional assessment is contained within a single while loop. The while loop continually reads off of a *queue* matrix containing voxel vertices. At the beginning of the algorithm, the queue matrix only contains one set of vertices: the original voxel. When the voxel is subdivided, its vertices are deleted from the queue and vertices for the eight subvoxels take its place. At this point the while loop begins again, working from the bottom to the top of the queue. Dividing a voxel adds 7 cells to the queue, whereas determining a voxel's opacity removes a cell. Over the course of the algorithm, the queue matrix will grow large with subvoxel vertices and then begin to regress back to zero. When the queue matrix has been exhausted the while loop breaks. At this point, voxels covering the entire defined region in the object space have been determined and the code is finished. The resulting final matrix contains a list of all voxels, their coordinates in the object space, and whether or not they lay on the object surface. The final step is to use this information to form a point cloud and ultimately determine a fitted surface for the object.

3.5 Final Product Manipulation

After the processing is complete, the final matrix of voxels can be used to visualize the object. However, due to the fact that voxels are of different sizes, displaying the model can become difficult. The most accurate way to do this is to display every opaque voxel as a cuboid with all six surfaces filled in with a color by using a MATLAB function such as *plotcube* [25]. The fact that there are hundreds of thousands of voxels, however, makes this an unmanageable task. Instead, the object can be modeled by plotting the centroids of

each voxel as a point cloud, a task that takes a significantly less amount of processing power. An example of this can be seen in Figure 17 below.

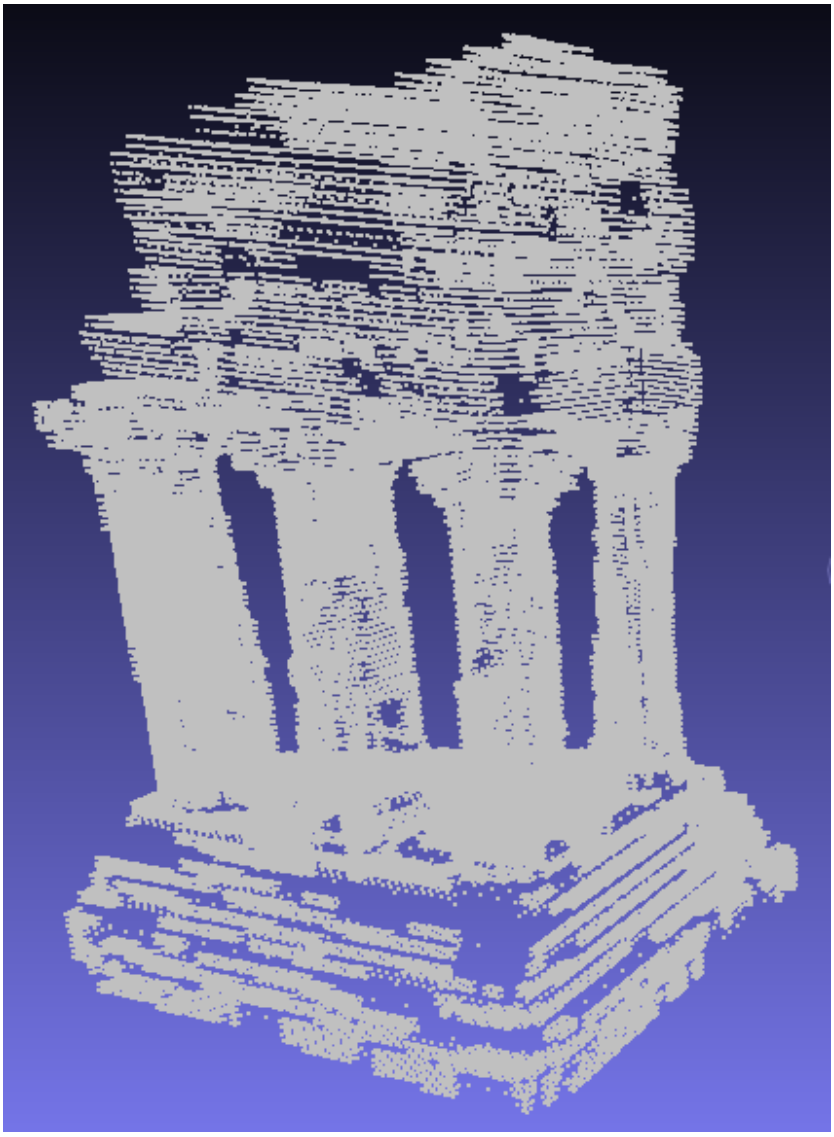

Figure 17 – 3D point cloud of temple model consisting of centroids of all voxels. Voxels determined using 10-pixel accuracy on TempleFull dataset:

As can be seen in the previous figure, there are quite a few holes. These holes, however, are due to visualization issues and not deficiencies in the algorithm. By only representing centroids on the plot, large voxels that were determined with little to no division are only represented by a single center point. This is the case in the above figure, easily seen for the large voxels that make up the base of the temple. In this case, modeling some voxels larger than others provides a visualization problem.

To alleviate this issue, a granularity adjustment is made. As opposed to traditional volumetric scene recovery that does projection and histogram calculations for each voxel at a very granular level, we can achieve the same granularity by simply dividing down the large voxels that we already have computed. It is known that all subvoxels will retain the same properties of the parent voxel, so the adjustment is a simple vertex division. To determine how many divisions are necessary, the volume of the smallest known voxel is calculated. As an example, the smallest voxels on the TempleFull dataset at 10-pixel accuracy were classified at the seventh layer of recursion. This means that the volume of the smallest voxel is $1/(8^6)$ of the first voxel (the bounding box of the object). All voxels larger than this size must then be divided until their subvoxels reach the same granularity. At this point, there will be 8^6 or 262,144 total voxels in the final matrix.

The algorithm that divides these voxels is separate from the original algorithm and takes only a few seconds to implement. This entire process is performed to alleviate the holes seen when displaying the original matrix. The same granularity is achieved as if we had performed traditional volumetric scene recovery, just without the computational burden.

The improved 3D model after the granularity adjustment can be seen in Figure 18. Note that all of the original holes have been filled in with very little additional computation.

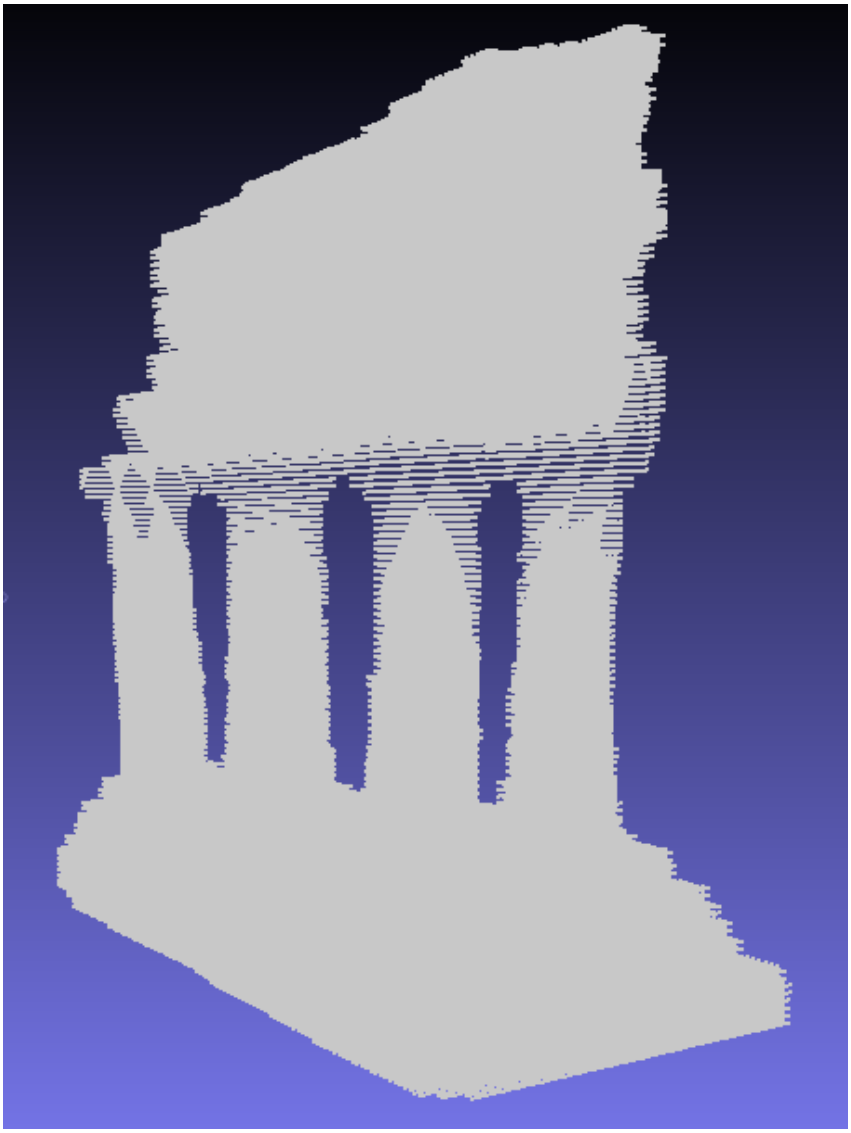

Figure 18 – 3D point cloud of temple model consisting of centroids of all voxels after division using granularity adjustment. Voxels determined using 10-pixel accuracy on TempleFull dataset:

As can be seen in the previous figure, fine features are not evident when viewing a point cloud. In order to better visualize these, it is necessary to fit a surface to the points. Many algorithms can be used to extract a three-dimensional polygonal mesh from a point cloud, many of which are included in an open-source software package called MeshLab [22, 23].

For the bounds of my research, I determined that using the "marching cubes" computer graphics algorithm would be most suitable for surface-fitting. The marching cubes algorithm was originally created and published in 1987 by Lorensen and Cline with the purpose of extracting and surface-fitting a polygonal mesh of a surface based on a three-dimensional scalar point cloud of voxels [20]. The algorithm works through the point cloud by taking eight neighbor locations at a time and forming them into an imaginary cube. Next, one of 256 possible polygon configurations between the eight points is selected based on a best-fit algorithm. Using this, vertices of the calculated polygon are linearly interpolated and saved as the surface for that particular cube. Eventually, the marching cubes algorithm works through all sets of eight and the surface is generated. Figure 19 shows a polygonal mesh surface calculated using a marching cubes algorithm set at 500-voxel grid resolution.

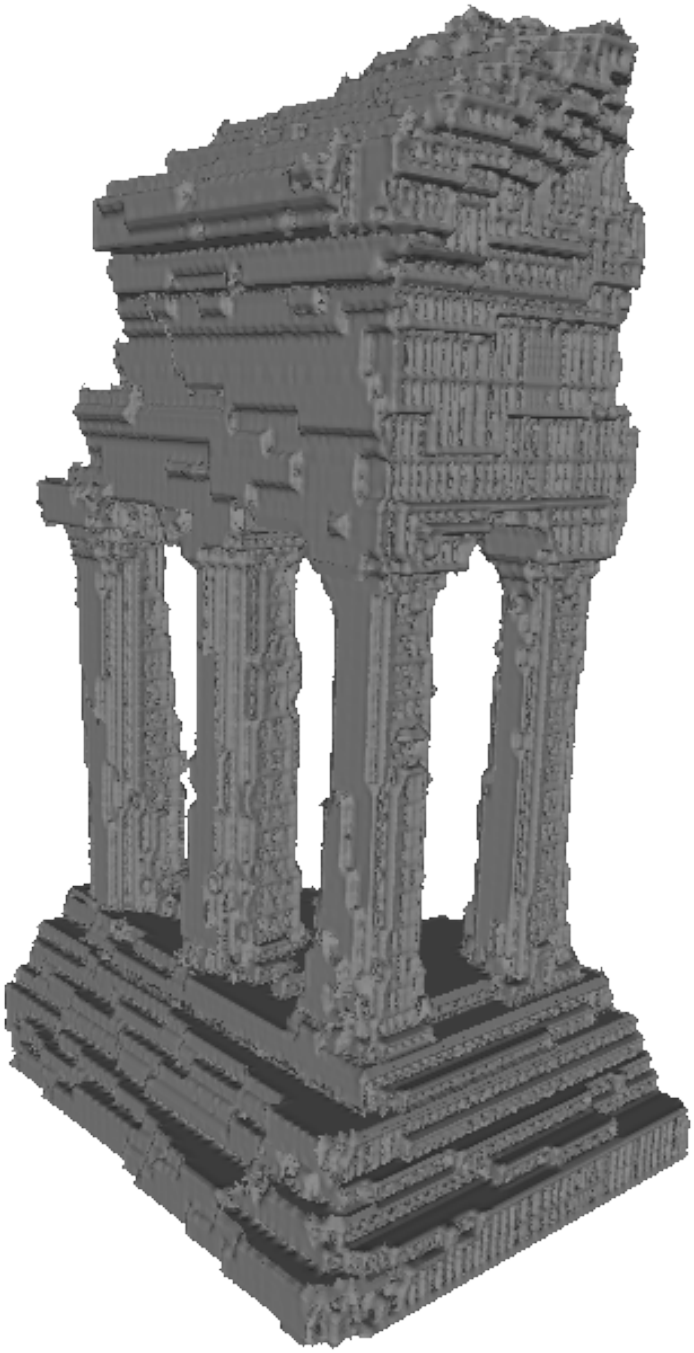

Figure 19 – Fitted surface of TempleFull, 10-pixel accuracy with granularity adjustment, created using a marching cubes algorithm at 500-voxel grid resolution:

Many different procedures can be implemented to further manipulate three-dimensional models. Different surface-fitting algorithms can be performed to produce different benefits. Smoothing can be applied to reduce edge roughness. Vertex and face colors can be adjusted and light sources can be added and rotated. Ultimately, the change of any arbitrary variable during processing or post-processing will produce a different resulting model. All in all, there is a seemingly infinite amount of setting combinations to achieve different model results. Despite the numerous combinations, quality and computational speed seem to always be close to inversely proportional. In a search for the optimal medium between quality and speed, a comparison is performed between some of the most notable adjustments that can be made in both processing and post-processing. The comparison results make up the final product of my algorithm and can be used to determine which variable settings should be used in different situations. The results of that comparison are displayed in Chapter 4.

CHAPTER 4 – RESULTS

4.1 – Processing Speeds

As discussed throughout this thesis, the main goal in implementing the voxel division algorithm is to reduce computational time by limiting the number of necessary computations. As such, a comparison of computation and duration statistics precede the display of visual quality results. It should be noted that the number of computations (i.e. number of voxels analyzed) takes precedence over processing time in comparing the speed of different methods. Processing times for traditional volumetric scene recovery are not available and those for my algorithm are varied. My processes were computed on multiple systems with varying capabilities and external loads, so processing times should be considered estimates and not comparison tools. For comparing methods, we conjecture that the number of computations is a more suitable measure. Each voxel's computation takes approximately the same amount of time to compute, independent of voxel size. With that in mind, it is possible to compare the number of computations needed in my results to the number of computations that traditional methods would require to create a model at an equivalent accuracy level. This comparison provides a suitable estimate of the time reduction achieved with the voxel division approach.

In traditional volumetric scene recovery methods, the number of voxel computations necessary is equal to the number of voxels in the grid. In the voxel division method the

number of computations is equal to $8^0+8^1+8^2+\ldots+8^n$ (where n=the number of divisions), assuming the worst case scenario where every voxel must be subdivided to the lowest subvoxel size. However, the nature of the voxel division method is that the lowest subvoxel size is not always necessary to classify a voxel. The actual number of computations in the voxel division method is equal to the above equation minus those subvoxels that are classified at a higher level. This number is based on multiple variables including dataset, histogram comparison, and pixel accuracy.

It should also be noted that my comparisons assume the traditional method of volumetric scene recovery is using the same dataset as my voxel division method. For example, the TempleFull and TempleSparseRing datasets (at the same accuracy level) require the same amount of voxel computations to compute the 3D model. However, since the TempleSparseRing dataset has to generate a significantly smaller number of histograms per voxel, this process runs much faster and in turn results in lower quality. In this case, the speed comparison is done between the computations needed for the traditional volumetric scene recovery method and those needed for the voxel division method, both using the TempleSparseRing dataset. To compare speed differences between individual datasets, computational time should be regarded rather than number of computations. With that said, comparisons of different variable combinations can be seen in Table 1. An explicit comparison of the primary models is covered in the next section.

Dataset	Sub-Dataset	Accuracy (pixels)	Comp. Time H:M:S	No. of Comp's	No. of Voxels	Traditional Comp's (8^n)	Estimated Processing Time Reduction
TEMPLE	FULL	100	5:03:47	31,057	27,175	262,144	88.2%
		10	18:53:37	125,761	110,041	2,097,152	94.0%
		3	99:26:15	502,713	439,874	16,777,216	97.0%
	RING	100	0:57:21	31,609	27,658	262,144	87.9%
		10	1:43:25	124,049	108,543	2,097,152	94.1%
		3	19:46:04	471,297	412,385	16,777,216	97.2%
	SPARSE RING	100	0:21:00	32,361	28,316	262,144	87.7%
		10	0:45:14	126,769	110,923	2,097,152	94.0%
		3	9:29:36	471,369	412,448	16,777,216	97.2%
DINO	FULL	100	6:59:23	37,753	33,034	262,144	85.6%
		10	22:48:12	130,482	114,281	2,097,152	93.8%
		3	108:21:22	536,435	484,287	16,777,216	96.8%
	RING	100	1:02:20	34,665	30,332	262,144	86.8%
		10	3:17:28	149,494	131,771	2,097,152	92.9%
		3	23:49:02	552,076	504,619	16,777,216	96.7%
	SPARSE RING	100	0:09:55	33,889	29,653	262,144	87.1%
		10	1:12:11	158,528	126,496	2,097,152	92.4%
		3	11:48:59	588,251	496,288	16,777,216	96.5%

Table 1 – Statistical comparison of computational time, number of computations, and number of final voxels in performing volumetric scene recovery using the voxel division algorithm on varying datasets and variables:

4.2 – Visual Results Comparison

There are several variables that when adjusted affect the outcome of the final 3D model significantly. As displayed in the table, these variables include changes in pixel accuracy, dataset, granularity, and marching cubes quality. This section contains a visual comparison and explanation of the results achieved when adjusting those variables.

The first comparison results from changes in pixel accuracy between 3, 10, and 100 pixels. As discussed in Section 3.4, the proposed algorithm contains a threshold that prohibits voxel division past a certain granularity level. This granularity level is determined by computing the average number of pixels contained within the projected polygonal convex hull on every stereo image. As such, a lower granularity level (e.g. 3 pixels) will allow the algorithm to further subdivide the voxels to obtain a higher granularity and resolution for the final model. As seen in Table 1, adjustments between pixel accuracy levels have significant impact on number of computations, number of voxels, and processing time. In turn, the images show that adjustments also have an inverse impact on model quality. Figure 20 shows the differences between 3, 10, and 100-pixel accuracy on the TempleFull dataset whose surface has been reconstructed using a 500-voxel resolution marching cubes algorithm.

Figure 20 – Qualitative comparison between completed models created with 100- (left), 10- (center), and 3- (right) pixel accuracies:

The following variable adjustment shows the difference in switching between the Full, Ring, and Sparse Ring sub-datasets. Each of these datasets contains full stereo coverage of the object in that no feature is unseen across all images. However, providing a denser set of stereo views increases histogram redundancy and provides higher quality results. For example, if a certain feature is shadowed from the viewpoint of the images of the TempleSparseRing dataset, the algorithm may have difficulty in determining the voxel's classification. However, when the same feature is viewed from many viewpoints (e.g. TempleFull) the impact of the shadow will be minimized and the voxel will be classified correctly. Similar to changing pixel accuracy, changes in the number of stereo images also impact processing speed since more projections and histogram computations are necessary. The computation and duration comparison of dataset adjustments can be seen in the table in the previous section. Figure 21 shows models created using the TempleFull, TempleRing, and TempleSparseRing datasets at 10-pixel accuracy whose surfaces have been reconstructed using a 500-voxel resolution marching cubes algorithm

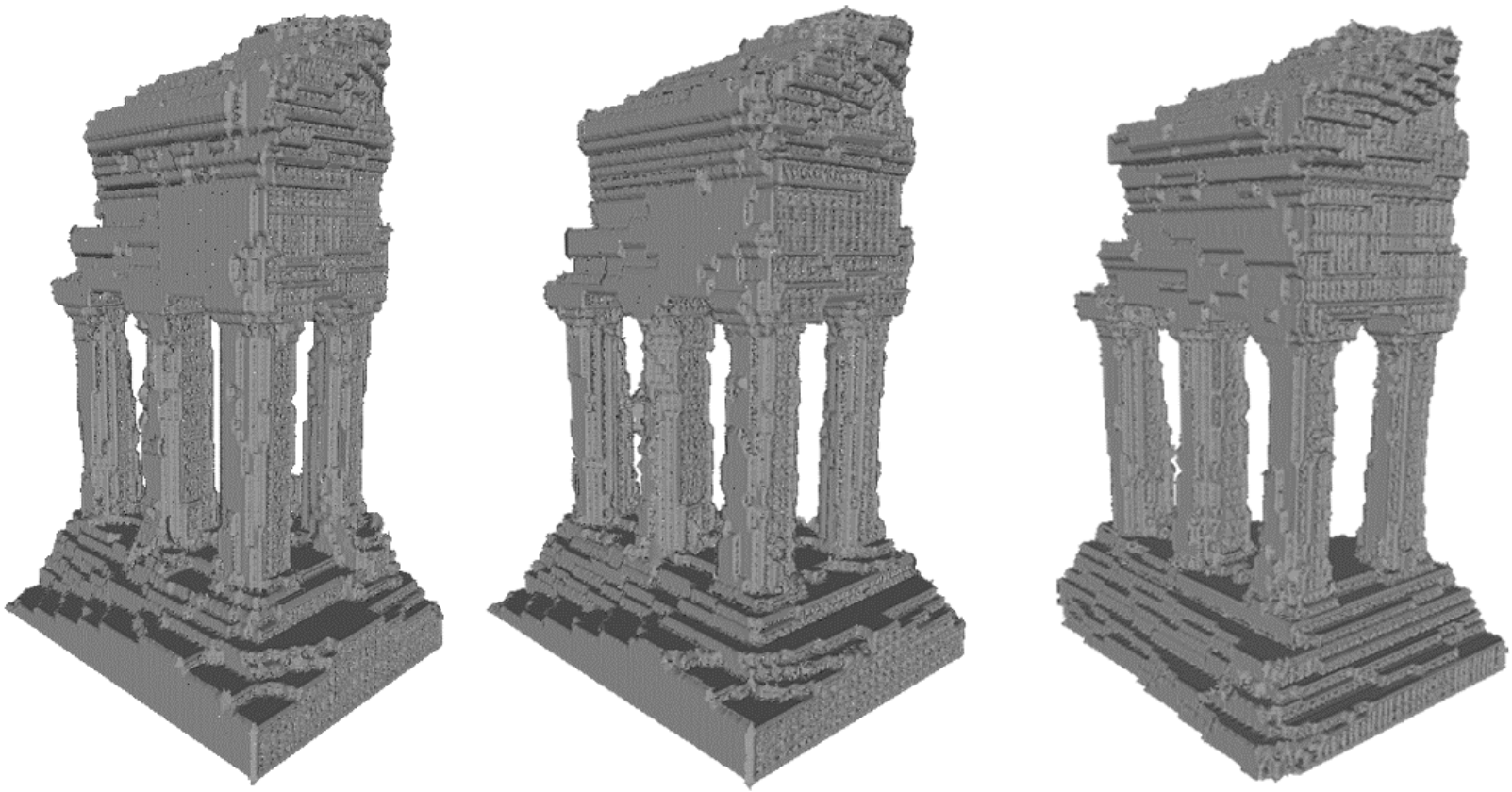

Figure 21 – Qualitative comparison between completed models created with TempleSparseRing(left, 16 views), TempleRing (center, 48 views), and TempleFull (right, 312 views) datasets:

The following two comparison models show the difference between surface-fitting results calculated using the direct point cloud output of my algorithm and those calculated using a point cloud that had been processed through the granularity adjustment.

Sparse point clouds directly result in unsatisfactory surface fitting results. A visual representation of the granular adjustment from sparse to dense point clouds can be seen in Section 3.5. The resulting models created from these point clouds can be seen in Figure 22. These models are created from the TempleFull dataset at 10-pixel accuracy with surface reconstruction done with a 500-voxel resolution marching cubes algorithm.

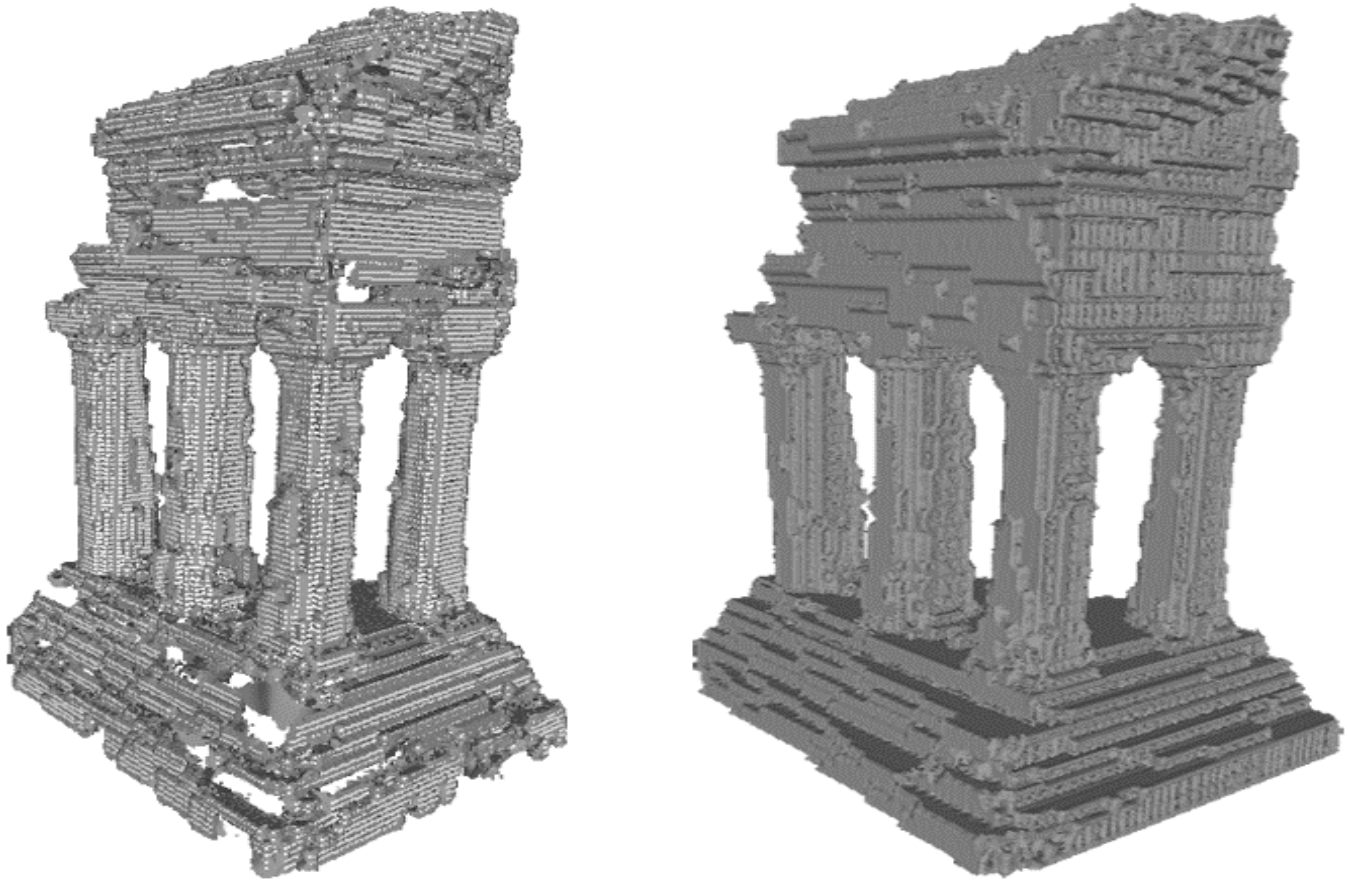

Figure 22 – Qualitative comparison between models created from a sparse (left) and granularity adjusted (right) point clouds:

The following comparison displays the results of adjusting the quality of the marching cubes algorithm. While not directly relevant to my research work, this comparison serves to show that adjustments made to surface fitting and other post-processing techniques are equally as important in generating desirable three-dimensional models. Figure 23 compares differences in surfaces created with 100, 200, and 500-voxel grid-resolution. Increased grid-resolution improves model quality. Unfortunately, models attempted above a 500-voxel grid-resolution crash the software and thus cannot be used for comparison. The models were created using the TempleFull dataset at 10-pixel accuracy.

Figure 23 – Comparison between models created using marching cubes surface fitting at 100- (left), 200- (center), and 500-voxel (right) grid-resolutions:

The final figure shows a comparison between my highest quality model and the ground truth model provided by Middlebury College. The ground-truth model, as seen in Section 3.1, represents a near-perfect model of the temple object captured using a laser-scanning camera. This model provides the best comparison tool for which the quality of my models can be assessed. It's likely that 3D reconstruction algorithms may never model an object as well as a laser-scanning camera, but the ground-truth model is beneficial in debugging problematic areas in the model's reconstruction. The best model produced by the proposed algorithm is seen below. The model uses the TempleFull dataset with 3-pixel accuracy and a surface modeled with a 800-voxel grid resolution marching cubes algorithm.

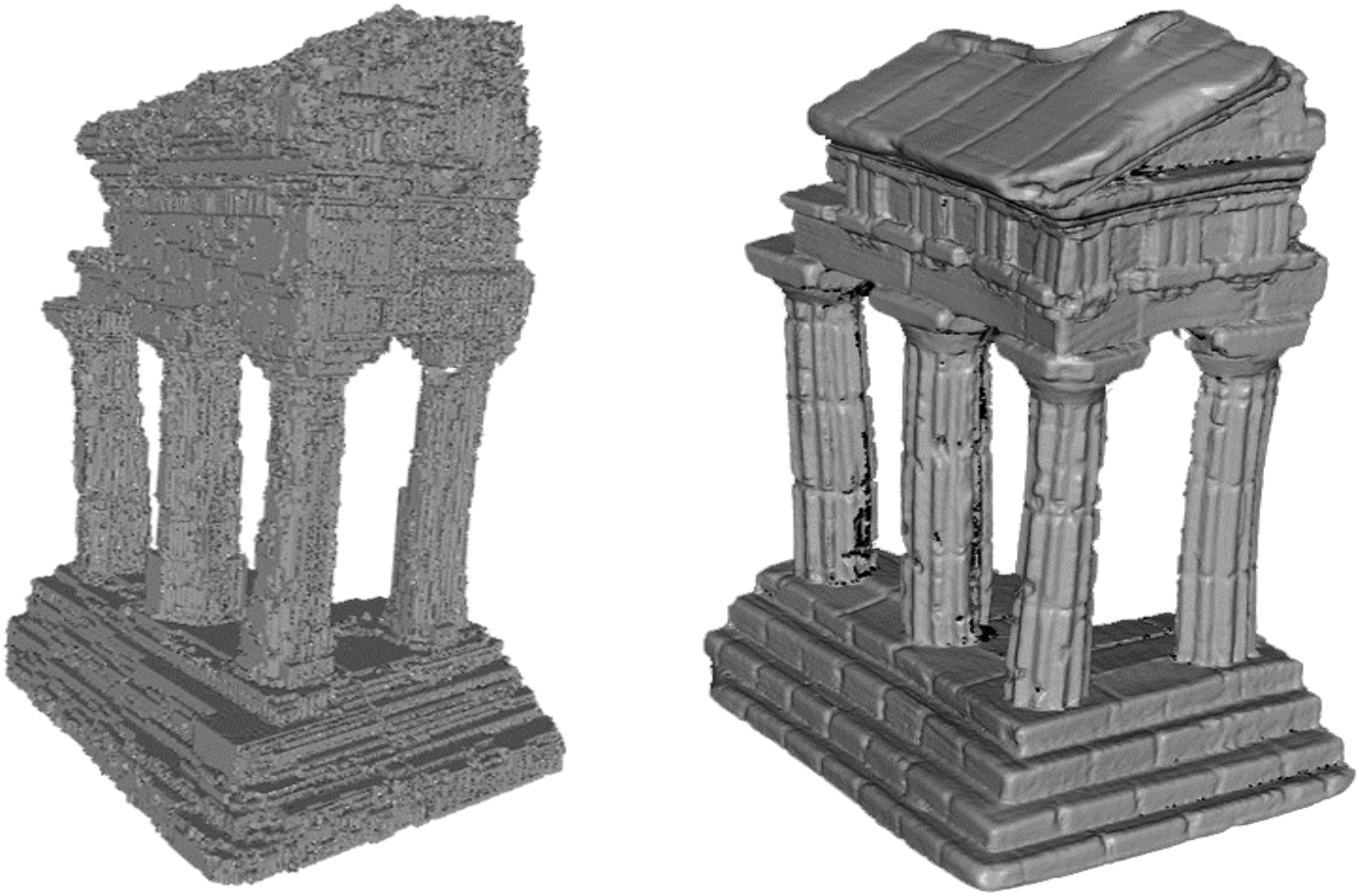

Figure 24 - Comparison of the highest quality model produced by the proposed algorithm to the ground truth model:

CHAPTER 5 – THE FUTURE

5.1 Practical Application

The success of researching and implementing an innovative technique such as the voxel division algorithm is dependent on the real-world applications for which the research can be used. Fortunately for my research, the use of software to create three-dimensional models is substantially increasing in many fields. With that increase in technology has also come a need for quality and efficiency. Many fields where this research is being incorporated need precise three-dimensional models, an issue addressed with the implementation of volumetric scene recovery. However, with this increased quality has come a need for improved computational speed, an issue tackled with the voxel division algorithm. The following practical applications are just a few examples of rapidly expanding areas where voxel division could have a significant impact.

In the field of photogrammetry and image understanding, volumetric scene recovery has the potential to become a key piece in automated surveillance. Currently, the majority of automated surveillance is done using single frame-capture cameras resulting in two-dimensional snapshots and videos. With the addition of a system of stereo cameras, volumetric scene recovery can be used to detect, track, and model movement on the ground [19]. In using three-dimensional imagery, interpreting personal actions and human interactions of tracked targets could become much simpler. In an area

such as automated surveillance swiftness is key, so the voxel division algorithm's ability to quickly render three-dimensional models could be of immense importance to this technology.

In popular culture, the use of three-dimensional applications is increasing exponentially [17]. From 3D movies to human models in video games, animators are constantly looking for a better method for generating three-dimensional products. In combat and sports video games, human renderings and movements are determined by applying sensors to an actor. The movements of the actor are captured and recreated within the game. Unfortunately, the sensors only provide a sparse set of movements. For example, a few sensors on the arm will likely model general arm movement, but may not pick up a turned wrist, wiggled finger, or a twist of the arm. If an efficient method of volumetric scene recovery were introduced, a dense set of points could be generated for the actor, thus capturing every movement to a much more accurate level. With a multi-billion dollar industry such as entertainment, a substantial quality and efficiency improvement could become the "next big thing in entertainment", thus greatly increasing revenue for production companies.

Another area where volumetric scene recovery could prove useful is in the use of innovative electronic-intensive-care-units (e-ICUs) [4]. In the past many hospitals have struggled to keep up with monitoring patients. With recent advancements these systems allow medical professionals to observe many patients simultaneously from one central observation center. In current setups, single two-dimensional video cameras are

monitoring patients. However, in order to more accurately recognize problems from the observation center, doctors are turning to three-dimensional techniques to be able to monitor patients from all angles. Eventually, a system of stereo cameras will be able to monitor patients in intensive-care units in full three-dimension. This system aims to reduce negligence in hospital settings by ensuring that patients in intensive-care units are constantly monitored. The addition of a more efficient algorithm, such as the voxel division approach proposed in this thesis, could improve problem recognition and response time from the observation center, likely saving lives in the process.

In a related field, a significant increase in the use of high-accuracy three-dimensional scene recovery is being seen in the study of biomechanics [30]. From medicine to sports science, understanding the underlying processes involved in human body movement has always been a difficult task. In recent years, scientists have used volumetric scene recovery to model the movement of the human skeleton by analyzing frames of captured stereo videos in hopes of acquiring a better understanding of the dynamics involved. With this knowledge, doctors and physical therapists could more effectively assist patients in recovering from injuries. As another example, professional sports trainers could improve the throwing motions of quarterbacks and pitchers, the swings of batters and golfers, and the flow of the shot of basketball players by simply modeling and analyzing their biomechanics and making adjustments as deemed necessary. The field of biomechanics is yet another area where a more efficient voxel division algorithm could have significant impact.

The voxel division method of performing volumetric scene recovery provides a solid foundation to improving the quality and efficiency of generating three-dimensional models. However, future research still remains before the algorithm can be used for more practical applications such as those described above. Some of this research will focus on optimizing the current algorithm to reduce problematic areas while other parts will focus on adding new features to improve quality. Some ideas for this future research can be seen in the following section.

5.2 Future Research

The ultimate goal for any volumetric scene recovery system is to eventually achieve a real-time system with the ability to create models instantaneously. Nearly all current uses for three-dimensional model reconstruction would benefit from instantaneous results, including many future uses not currently possible. While not its own explicit volumetric method, the voxel division algorithm is a system that can be applied to existing and future methods to greatly improve speed. With continued research, the voxel division method has the potential to become the basis for this proposed real-time system.

Before being transported to a real-time system, the voxel division approach would first need to be converted to a faster compiler. To benefit from built-in functions, the current code is written entirely in MATLAB. However, converting the code and built-in functions to a compiling language such as C++ would increase speed even more. In addition to the code conversion, camera auto-calibration software would be a necessity for the system. In a real-time system it's assumed that the object being reconstructed or

the stereo cameras taking the images may not be static. Since accurate camera parameters are necessary for volumetric scene recovery methods, a real-time system would have to be implemented to automatically calibrate the cameras as they or the object move. To go along with this, another significant need would be camera network synchronization. Assuming that each camera does not have it's own computer, complex synchronization would be necessary to tie together the cameras, their resulting images, and all necessary parameters. The source code, auto-calibration software, and camera network synchronization would form the basic framework for a real-time volumetric scene recovery system.

In addition to the proposed framework, a few issues in the voxel division algorithm were discovered during my research. It's possible that there are more underlying problems, but that is to be expected in the early stages of research. The first issue recognized is the inability to model crevices due to lack of stereo views. As seen in Figure 25, the interior corner of the temple was incorrectly labeled as opaque, thus resulting in a large chunk of artificial object. This issue arises because of the position of the voxels in three-dimensional space. From a human standpoint it is easy for us to identify that this area is empty, but the algorithm is not as intelligent. The projection to these voxels from all viewpoints is blocked by other sections of the object when looking at the scene in two-dimensions. To manually fix this problem, a camera would need a viewpoint from inside the temple looking directly vertical (up the *z*-axis) to correctly identify that area as empty. Alternatively, existing algorithms such as visibility checks and shadow analysis could potentially alleviate this issue.

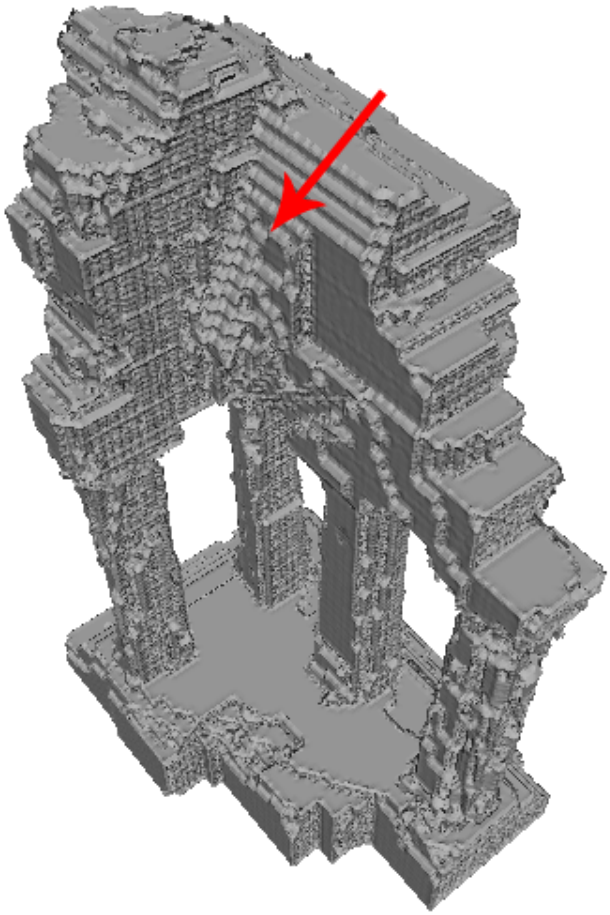

Figure 25 – Completed Temple model showing problematic region in corner crevice as outlined by the red arrow:

Another problem with the voxel division method arises in instances where the model being generated requires a high granularity for an accurate representation. Imagine attempting a three-dimensional reconstruction of a chain-link fence or a cheese grater where the object is comprised of many holes (i.e. many changes between background and foreground color). In a case like this, a high granularity reconstruction such as in the

traditional method would need to be performed to accurately model the object. However, the voxel division algorithm would take many computations just to get to the highly granular level. Whereas the traditional method requires 8^n computations, the voxel division method would have to perform $8^0+8^1+8^2+. . .+8^n$. The entire premise of the voxel division method requires that there are large areas in the object space that can be generalized, thus saving time from the granular traditional method. If these areas are not present, this method will likely be slower than the traditional method. Further research could determine at what granularity the processing time paths of the two methods cross and could institute a condition for efficiency. In that case, if a very granular scene is detected the program would automatically perform a traditional recovery rather than going through the recursive loops of voxel division. With an object like this, the voxel division method ultimately would not save time, but it wouldn't lose it either.

In addition to the previous issue improvements, there are also a few optional features that could be added to improve the quality of reconstructed models. The first optional feature would be the addition of color. While the premise of reconstructing three-dimensional models is in determining which voxels are part of the model and which are not (i.e. transparent vs. opaque), the addition of texture mapping can provide more eye-appealing models. However, this task is complex and would significantly increase the processing time of the reconstruction. Instead of performing a single histogram check of background-to-foreground color ratios, a check would be done across red, green, and blue histograms. The time it takes to determine if a voxel is transparent or opaque would remain the same, but further division would likely be necessary to separate and discern

different colors. Unless a more efficient algorithm for color determination was introduced, this would likely defeat the purpose of the voxel division techniques.

In addition to texture mapping, certain preprocessing techniques could allow for increased quality and efficiency. The main aspect of this would be a feature that could automatically detect the object being modeled. Whereas in my dataset I am given a bounding box that defines the bounds of the object in three-dimensional space, a more practical application of this method would need to automatically detect this. If this were ignored, the entire scene would be modeled (including background), thus resulting in unnecessary calculations. In addition to separating background and foreground, this detection would also assist in the histogram ratio threshold. In the current dataset, a spike in the histogram defines the switch between background and foreground colors. This threshold is used in the algorithm to determine if voxels are opaque (foreground) or transparent (background). However, the threshold differs on every object scene and thus needs to be manually entered to achieve correct background-to-foreground ratios. The object detection feature would secondarily determine the histogram ranges for background and foreground and thus would make this part of the algorithm automatic.

These improvements provide only a sub-sampling of the research that can be done to further improve the voxel division algorithm. Being in its infancy, the new algorithm has a lot of potential to become a new standard in the use of volumetric scene recovery. Eventually a real-time volumetric scene recovery system will be a reality and I'm hopeful that the offspring of the voxel division algorithm will be a part of it.

5.3 Summary

The goal of this thesis was to create and implement a more efficient algorithm for performing high quality three-dimensional volumetric scene recovery. Through my research I've come to the conclusion that this approach is in fact faster than traditional granular methods in almost all scenarios. All else being equal, computational time reductions of up to 97% were achieved with unwavering quality in comparison to models created using traditional granular algorithms. In one case, an entire three-dimensional model was reconstructed in less than 10 minutes with acceptable quality. These results strongly support my hypothesis that the voxel division algorithm is superior to traditional methods.

Much research is in progress on volumetric scene recovery techniques, all of which focuses on improving the quality of results with little regard for computational time. The voxel division algorithm benefits from the fact that it is compatible and can be transported to any current volumetric scene recovery technique to improve computational speed. Hopefully this increase in efficiency allows other researchers to better improve their methods of generating high quality models. In addition to model quality improvements, further improvements to the voxel division algorithm will produce even faster computational speeds. When perfected, the combination of improved quality and speed will create the foundation for the practical use of volumetric scene recovery in the advancement of many fields.

REFERENCES

[1] Bartoli, Adrien, and Peter Sturm. "Structure-From-Motion Using Lines: Representation, Triangulation and Bundle Adjustment." Web.

[2] Beardsley, P.A., A. Zisserman, and D.W. Murray, “Navigation Using Affine Structure from Motion”, Proc. of 3rd European Conference on Computer Vision, Stockholm, 1994. Springer Verlag, 1994.

[3] Bebis, G. "Stereo Reconstruction Problems." Department of Computer Science, University of Nevada. Web. <http://www.cse.unr.edu/~bebis/CS791E/Notes/StereoReconstruction.pdf>.

[4] Breslow, M.J., B.A. Rosenfeld, M. Doerfler, G. Burke, G. Yates, D.J. Stone, P. Tomaszewicz, R. Hochman, and D.W. Plocher. “Effect of a Multi-Site Intensive Care Unit Telemedicine Program on Clinical and Economic Outcomes: An Alternative Paradigm for Intensivist Staffing,” in *Crit Care Med 2004*, 32:31-38

[5] Bykat, A., "Convex Hull of a Finite Set of Points in Two Dimensions", Info. Proc. Letters 7, 296-298 (1978)

[6] Cormen, Thomas H., Charles E. Leiserson, and Robert L. Rivest. *Introduction to Algorithms*. Cambridge, MA: MIT, 2001. Print.

[7] "convhulln." *MathWorks - MATLAB and Simulink for Technical Computing*. Web. <http://www.mathworks.com/help/techdoc/ref/convhulln.html>.

[8] Dow, Mark. "Point Cloud to Mesh." *UofO Lewis Center for Neuroimaging*. Web. <http://lcni.uoregon.edu/~mark/Projects/Brain_casting/Point_cloud_to_mesh.html>.

[9] Dyer, C., “Volumetric Scene Reconstruction from Multiple Views,” in Foundations of Image Understanding, pp. 469–489, Kluwer, 2001.

[10] Eddy, W., "A New Convex Hull Algorithm for Planar Sets", ACM Trans. Math. Software 3(4), 398-403 (1977)

[11] “find.” *MathWorks – MATLAB and Simulink for Technical Computing.* Web. <http://www.mathworks.com/help/techdoc/ref/find.html>.

[12] Harris, C. and M. Stephens, “A Combined Corner and Edge Detector,” in 4th Alvey Vision Conference, pp. 147–151, 1988

[13] Hartley, Richard. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge: Cambridge UP, 2004. Print.

[14] Image adapted from the Wikimedia Commons file "Image:ConvexHull.png” <http://commons.wikimedia.org/wiki/Image:ConvexHull.png>.

[15] Image adapted from the Wikimedia Commons file "Image:Flemish_Bond.jpg” <http://commons.wikimedia.org/wiki/File:Flemish_Bond.jpg>.

[16] Image adapted from the Wikimedia Commons file "Image:Tree_Leaves.JPG” <http://commons.wikimedia.org/wiki/File:Tree_Leaves.JPG>.

[17] Izquierdo, E. and J.R. Ohm, “Image-Based Rendering and 3D Modeling: A Complete Framework,” in *Signal Processing: Image Communication*, Vol. 15, No. 10, August 2000, pp. 817-858

[18] Jain, Varun, and Xiaoxing Li. "Point Matching Methods: Survey and Comparison." Print.

[19] Javed, Omar, and Mubarak Shah. "Tracking and Object Classification for Automated Surveillance." Ed. Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen.*Computer Vision - ECCV 2002*. Vol. 2353. Heidelberg: Springer Berlin, 2006. 439-43. Print.

[20] Lorensen, William E. and Harvey E. Cline, “Marching Cubes: A High-Resolution 3D Surface Construction Algorithm”, in Computer Graphics, Vol. 21, Nr. 4, July 1987

[21] Lowe, D., “Distinctive Image Features from Scale-Invariant Keypoints,” Int. Jrn. on Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.

[22] "Meshing Point Clouds." *MeshLab Stuff*. 7 Sept. 2009. Web. <http://meshlabstuff.blogspot.com/2009/09/meshing-point-clouds.html>.

[23] *MeshLab*. 3D-CoForm Project. Web. <http://meshlab.sourceforge.net/>.

[24] *Multi-View Datasets*. Middlebury College Stereo Vision, 20 July 2007. Web. <http://vision.middlebury.edu/mview/data/>.

[25] "plotcube." *MathWorks - MATLAB and Simulink for Technical Computing*. Web. <http://www.mathworks.com/matlabcentral/fileexchange/15161-plotcube>.

[26] “poly2mask.” *MathWorks – MATLAB and Simulink for Technical Computing.* Web. <http://www.mathworks.com/help/toolbox/images/ref/poly2mask>.

[27] Seitz, S. M., B. Curless, J. Diebel, D. Scharstein, and R. S. Szeliski. "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms." Proc. of CVPR. Vol. 1. 2006. 519-26. Print.

[28] Slabaugh, G., B. Culbertson, T. Malzbender, and R. Schafer, “A Survey of Methods for Volumetric Scene Reconstruction from Photographs,” VolumeGraphics, vol. 1, pp. 81–100, 2001.

[29] Stockman, George. "Point/Feature Matching Methods." Proc. of CVPR. 2004. Print.

[30] Tyson, J., T. Schmidt, and K. Galanulis, “Biomechanics Deformation and Strain Measurements with 3D Image Correlation Photogrammetry,” in *Experimental Techniques,* 26:39-42, 2002

APPENDIX

Matlab Source Code

Frame.m

Frame.m is the most outside function in my algorithm. This function takes user input, inputs data from the dataset files, calls the Go.m function, and saves the FinalMatrix output file.

```
%Frame.m
tic
clear all
clc
close all

%User Input----------------------------------------------------------------
   StartTime='4:38 PM'; %Start time of process
   Quality=3; %Pixel-accuracy for reconstruction
   Dataset='dinoSR'; %Dataset to be used
%--------------------------------------------------------------------------


%Orientation Parameters----------------------------------------------------
   [length P]=BuildP(Dataset);
   %This function calls the camera orientation parameters and builds
   %the projection matrix.
   %It returns length (number of images) and P (projection matrix).

   %Define bounding box for selected dataset:
   if strcmp(Dataset,'temple')==1
   x1=-.054568; %These are the bounds of the boundary box
   y1=.001728;
   z1=-.042945;
   x2=.047855;
   y2=.161892;
   z2=.032236;
   BlackThreshold=3; %Define histogram threshold (switch from
   %background to foreground colors)
   end
   if strcmp(Dataset,'templeR')==1
   x1=-.023121;
   y1=-.038009;
   z1=-.091940;
   x2=.078626;
   y2=.121636
```

```
    z2=-.012951;
    BlackThreshold=3;
    end
    if strcmp(Dataset,'templeSR')==1
    x1=-0.073568;
    y1=0.021728;
    z1=-0.012445;
    x2=0.028855;
    y2=0.181892;
    z2=0.062736;
    BlackThreshold=3;
    end
    if strcmp(Dataset,'dino')==1
    x1=-.041897;
    y1=.001126;
    z1=-.037845;
    x2=.030897;
    y2=.088227;
    z2=.035495;
    BlackThreshold=2;
    end
    if strcmp(Dataset,'dinoR')==1
    x1=-.021897;
    y1=.021126;
    z1=-.017845;
    x2=.050897;
    y2=.108227;
    z2=.055495;
    BlackThreshold=2;
    end
    if strcmp(Dataset,'dinoSR')==1
    x1=-.061897;
    y1=-.018874;
    z1=-.057545;
    x2=.010897;
    y2=.068227;
    z2=.015495;
    BlackThreshold=2;
    end
%-----------------------------------------------------------------------

StartAndEnd=[x1,y1,z1,x2,y2,z2];
[BoundingBox]=Bound(StartAndEnd);
%This function builds a (4,8) matrix that defines the coordinates of the bounding box around the scene.
%There are 8 coordinates, 1 per column. Each column has the format
%(x;y;z;1)

FinalMatrix=zeros(1,11);%[X1,Y1,Z1,X2,Y2,Z2,0/64,Condition,VoxelVolume)
index=1;
ISize=size(imread(strcat('Z:\Thesis\Dataset\',Dataset,'\Grayscale\',Dataset,'1.png'),'png'));
%Define image size

[FinalMatrix Queue Slot ticker]=Go(BoundingBox,length,P, FinalMatrix, index, Dataset, ISize, MVolume ,
```

```
Quality, BlackThreshold);
%This is the function containing the while loop performing voxel division

save(strcat('Final',num2str(Quality),Dataset),'FinalMatrix','Queue','Slot','ticker','X');
```

Go.m

Go.m comprises the main body of the code. This function calls many sub-functions to perform tasks for voxel analysis. This function contains the while loop that reads off of the queue matrix. When the queue matrix is emptied, Go.m returns the FinalMatrix variable to Frame.m

```
%Go.m
function [FinalMatrix Queue slot ticker]=Go(BoundingBox,length,P, FinalMatrix, index, Dataset, ISize,
MVolume, Quality, BlackThreshold)
Queue=BoundingBox;
slot=1; %Index of current vertices being analyzed in Queue matrix
IMain=cell(length,1);

parfor k=1:length %Read in all stereo images into IMain cells
   IMain{k}=imread(strcat('Z:\Thesis\Dataset\',Dataset,'\Grayscale\',Dataset,int2str(k),'.png'),'png');
end

while Queue(1:4,1:8) ~= zeros(4,8) %While queue matrix is not empty

   ticker=ticker+1; %Keeps track of # of computations
   BoundingBox=Queue(slot:(slot+3),1:8); %Define current voxel in Queue


   %This converts the 3-D coordinates to 2-D coordinates for every image.
   %It returns TwoDimScaled, the 2-d matrix after x and y
   %have been scaled by k.
   TwoDimScaled=zeros(3,8,length);%Create matrix that will hold 2-D coordinates of each corner. It is
(3,8,:) where : is the # of images. The form for each column is [x,y,k]', and each column corresponds to the
3-D points in the same order as listed above
   for k=1:length
      TwoDimScaled(:,:,k)=P(:,:,k)*BoundingBox;%Find 2-D coordinates using P and 3-D coordinate
matrix
      for l=1:8
         for m=1:2
            TwoDimScaled(m,l,k)=TwoDimScaled(m,l,k)/TwoDimScaled(3,l,k);
         end
      end
   end


   %Convex Hull Determination----------------------------------------
   K=cell(length,1);
   TwoDimScaled=TwoDimScaled(1:2,:,:);
   for p=1:length
      TwoDim=TwoDimScaled(:,:,p)';
```

```
    TwoDim2=TwoDim(:,2:-1:1);
    J=convhulln(TwoDim2);
    [f g]=size(J);
    L=round(cast(((TwoDim(J(1:f),1:2))'),'double'));
    L(L<=0)=1;
    for i=1:f
      if L(1,i)>ISize(2)
        L(1,i)=ISize(2);
      end
      if L(2,i)>ISize(1)
        L(2,i)=ISize(1);
      end
    end
    K{p}=L;
  end

  [LargestRatio AvgPix AvgRatio]=Histogram(length, K, ISize, IMain, BlackThreshold);
  %This function determines polygonal mask, histograms, and black level
  %ratios for every image. The maximum black level ratio is returned
  %-------------------------------------------------------------------



  %Condition Assessment-----------------------------------------------
  if AvgPix<=Quality %Condition 3 - Below quality limit
    if AvgRatio <= .5 %Voxel labeled opaque
      FinalMatrix(index,:)=[StartAndEnd,0, 1, Volume, FinalMatrix(index-
1,10)+((Volume/MVolume)*100)];
    else %Voxel labeled transparent
      FinalMatrix(index,:)=[StartAndEnd,64, 1, Volume, FinalMatrix(index-
1,10)+((Volume/MVolume)*100)];
    end
    index=index+1; %Index of FinalMatrix increases by 1
    Queue(slot:slot+3,:)=zeros(4,8); %Zero out voxel slot in Queue
    if slot==1; %Queue empty
      return
    end
    slot=slot-4; %Change Queue slot
  else
    %Condition 1 - Histogram is over 90% black
    if LargestRatio >= .90
      FinalMatrix(index,:)=[StartAndEnd,0, 2, Volume, ((Volume/MVolume)*100)];
      index=index+1; %Index of FinalMatrix increases by 1
      Queue(slot:slot+3,:)=zeros(4,8); %Zero out voxel slot in Queue
      if slot==1; %Queue empty
        return
      end
      slot=slot-4; %Change Queue slot
    else
    %Condition 4 - Between 10% and 90% Black (subdivide)
      if LargestRatio > .10
        [BoundingBox]=subdivide(StartAndEnd);
        %This function takes the 8 voxel vertices and returns the
        %64 sub-voxel vertices
```

```
            Queue(slot:slot+31,1:8)=BoundingBox; %Fill Queue with new vertices
            slot=slot+28; %Change queue slot
         else
         %Condition 2 - Histogram is <10% black
            FinalMatrix(index,:)=[StartAndEnd,64, 3, Volume, FinalMatrix(index-
1,10)+((Volume/MVolume)*100)];
            index=index+1; %Index of FinalMatrix increases by 1
            Queue(slot:slot+3,:)=zeros(4,8); %Zero out voxel slot in Queue
            if slot==1; %Queue empty
               return
            end
            slot=slot-4; %Change Queue slot
         end
      end

   end
   %---------------------------------------------------------------
end
```

Histogram.m

Histogram.m is called by Go.m to perform the polygon mask, histogram creation, and ratio determination. The maximum black level ratio is returned to Go.m to fulfill one of the four conditions.

```
%Histogram.m
function[MaxBlack, AvgPix]=Histogram(length, K, ISize, IMain, BlackThreshold)
MaxBlack=0;
Pixels=0;
for i = 1:length %For each stereo image
   P=K{i};
   minx = round(min(P(1,:)));
   miny = round(min(P(2,:)));
   maxx = round(max(P(1,:)));
   maxy = round(max(P(2,:)));

   %Create polygon mask
   mask = poly2mask(P(1,:)-minx+1,P(2,:)-miny+1,maxy-miny+1,maxx-minx+1);
   Pic=IMain{i};
   sizeM = size(mask);
   F=Pic(miny:(miny+sizeM(1))-1,minx:(minx+sizeM(2))-1);
   %Find where on image the mask=1
   Histograms=imhist(F(mask==1),64);

   Ratio=sum(Histograms(1:BlackThreshold))/sum(Histograms);
   if Ratio > MaxBlack
      MaxBlack=Ratio;
   end
   Pixels=Pixels+sum(Histograms);
end
AvgPix=Pixels/length; %To compare with pixel-accuracy limit
```

MakeGranular.m

MakeGranular.m is the post-processing function that reads in the FinalMatrix and divides all voxels to the same level as the smallest voxel contained in the matrix. This is performed for better visualization of the point cloud and better reconstruction of the object using surface-fitting techniques.

```
%MakeGranular.m
function [FinalMatrixGRAN] = MakeGranular(FinalMatrix)
n=round((sum(FinalMatrix(:,9))/min(FinalMatrix(:,9)))^(1/8));
FinalMatrix=FinalMatrix(:,1:9);
[m,n]=size(FinalMatrix);
index=1;
M=min(FinalMatrix(:,9));
for i=1:(n-1)
   tic
[m,n]=size(FinalMatrix);
index=m+1;
P=FinalMatrix(:,9);
X=find(P>2*M);
U=length(X);
FinalMatrix(m+1:m+U*8,9)=0;
for j=1:U
      x1=FinalMatrix(X(j),1);
      y1=FinalMatrix(X(j),2);
      z1=FinalMatrix(X(j),3);
      x2=FinalMatrix(X(j),4);
      y2=FinalMatrix(X(j),5);
      z2=FinalMatrix(X(j),6);
      x3=(x1+x2)/2;
      y3=(y1+y2)/2;
      z3=(z1+z2)/2;
      V=FinalMatrix(X(j),9)/8;
      FinalMatrix(index,1:9)=[x1,y1,z1,x3,y3,z3,FinalMatrix(X(j),7),0,V];
   FinalMatrix(index+1,1:9)=[x1,y1,z3,x3,y3,z2,FinalMatrix(X(j),7),0,V];
      FinalMatrix(index+2,1:9)=[x3,y1,z1,x2,y3,z3,FinalMatrix(X(j),7),0,V];
      FinalMatrix(index+3,1:9)=[x3,y1,z3,x2,y3,z2,FinalMatrix(X(j),7),0,V];
      FinalMatrix(index+4,1:9)=[x1,y3,z1,x3,y2,z3,FinalMatrix(X(j),7),0,V];
      FinalMatrix(index+5,1:9)=[x1,y3,z3,x3,y2,z2,FinalMatrix(X(j),7),0,V];
      FinalMatrix(index+6,1:9)=[x3,y3,z1,x2,y2,z3,FinalMatrix(X(j),7),0,V];
      FinalMatrix(index+7,1:9)=[x3,y3,z3,x2,y2,z2,FinalMatrix(X(j),7),0,V];
      index=index+8;
      FinalMatrix(X(j),:)=0;
end
B=sum(FinalMatrix,2);
a=find(B~=0);
FinalMatrix=FinalMatrix(a,:);
toc
end
save('FinalMatrixGRAN','FinalMatrix');
```